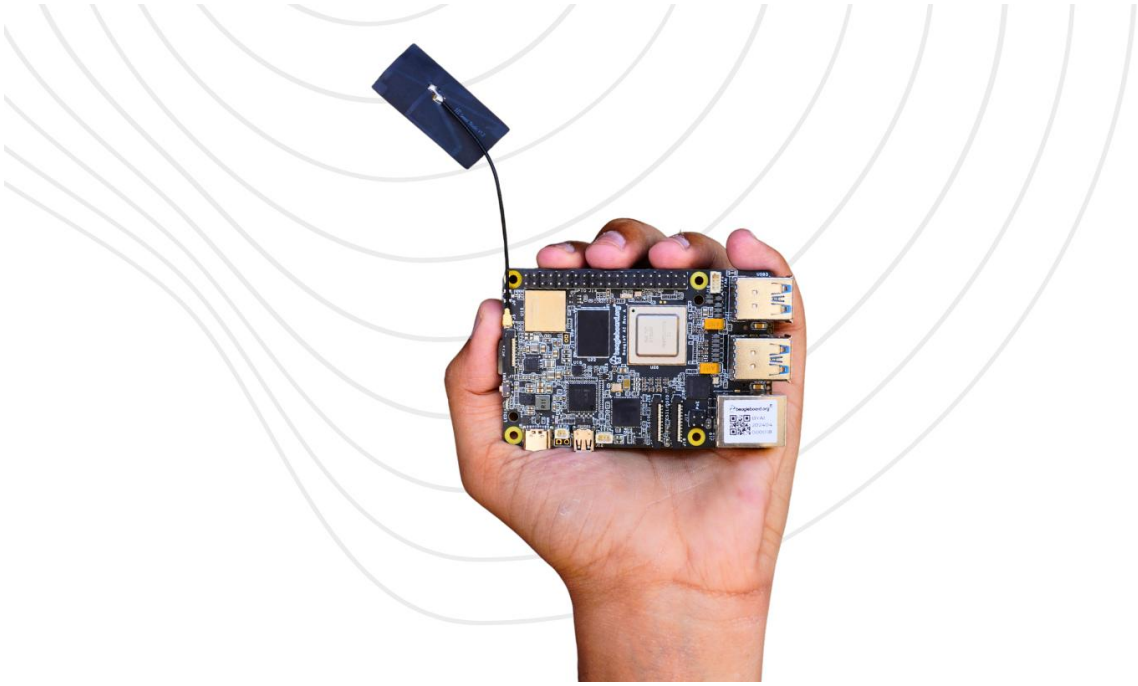
BeagleY-AI

BeagleBoard.org Foundation
Sep 19, 2024

Table of contents

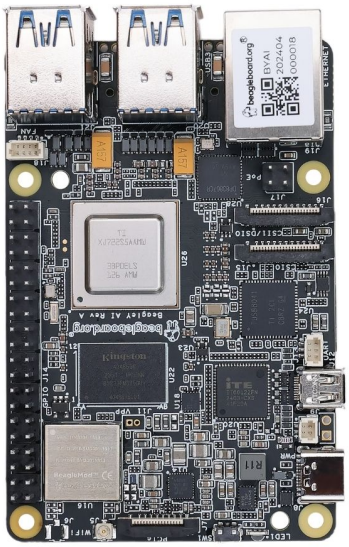
BeagleY-AI is an open-source single board computer based on the Texas Instruments AM67A Arm-based vision processor.

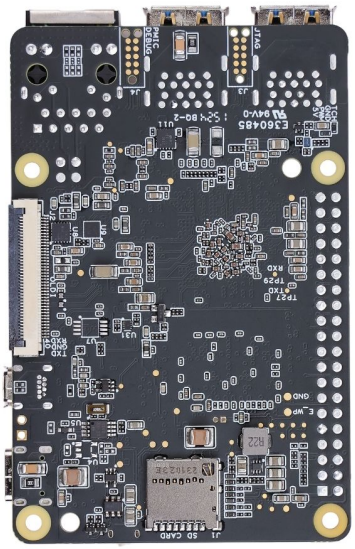


Chapter 1

Introduction

BeagleY-AI is an open-source single board computer designed for edge AI applications.





1.1 Detailed overview

BeagleY-AI is based on the Texas Instruments AM67A Arm-based vision processor. It features a quad-core 64-bit Arm®Cortex®-A53 CPU subsystem at 1.4GHz, Dual general-purpose C7x DSP with Matrix Multiply Accelerator (MMA) capable of 4 TOPs each, Arm Cortex-R5 subsystem for low-latency I/O and control, a 50 GFlop GPU, video and vision accelerators, and other specialized processing capability.

Table 1.1: BeagleY-AI features

Feature	Description
Processor	Texas Instruments AM67A, Quad 64-bit Arm® Cortex®-A53 @1.4 GHz, multiple cores including Arm/GPU processors, DSP, and vision/deep learning accelerators
RAM	4GB LPDDR4
Wi-Fi	Beagleboard BM3301, 802.11ax Wi-Fi
Bluetooth	Bluetooth Low Energy 5.4 (BLE)
USB Ports	4 x USB 3.0 TypeA ports supporting simultaneous 5Gbps operation, 1 x USB 2.0 TypeC, supports USB 2.0 device mode
Ethernet	Gigabit Ethernet, with PoE+ support (requires separate PoE HAT)
Camera/Display	2 x 4-lane MIPI camera connector (one connector muxed with DSI capability)
Display Output	1 x HDMI display, 1 x OLDI display, 1 x DSI MIPI Display
Real-time Clock (RTC)	Supports external coin-cell battery for power failure time retention
Debug UART	1 x 3-pin debug UART
Power	5V/3A DC power via USB-C
Power Button	On/Off included
PCIe Interface	PCI-Express® Gen3 x 1 interface for fast peripherals (requires separate M.2 HAT or other adapter)
Expansion Connector	40-pin header
Fan connector	1 x 4-pin fan connector, supports PWM control and fan speed measurement
Storage	microSD card slot with UHS-1 support
Tag Connect	1 x JTAG, 1 x External PMIC programming port

1.1.1 AM67A SoC

The AM67A scalable processor family is based on the evolutionary Jacinto™ 7 architecture, targeted at Smart Vision Camera and General Compute applications and built on extensive market knowledge accumulated over a decade of TI's leadership in the Vision processor market. The AM67A family is built for a broad set of cost-sensitive high performance compute applications in Factory Automation, Building Automation, and other markets.

Some Applications include:

- Human Machine Interface (HMI)
- Hospital patient monitoring
- Industrial PC
- Building security system
- Off-highway vehicle
- Test and measurement
- Energy storage systems
- Video Surveillance
- Machine Vision
- Industrial mobile robot (AGV/AMR)
- Front camera systems

The AM67A provides high performance compute technology for both traditional and deep learning algorithms at industry leading power/performance ratios with a high level of system integration to enable scalability and lower costs for advanced vision camera applications. Key cores include the latest Arm and GPU processors for general compute, next generation DSP with scalar and vector cores, dedicated deep learning and traditional algorithm accelerators, an integrated next generation imaging subsystem (ISP), video codec, and MCU cores. All protected by industrial-grade security hardware accelerators.

Tip: For more information about AM67A SoC you can checkout https://www.ti.com/product/AM67A

1.2 Board components location

1.2.1 Front components

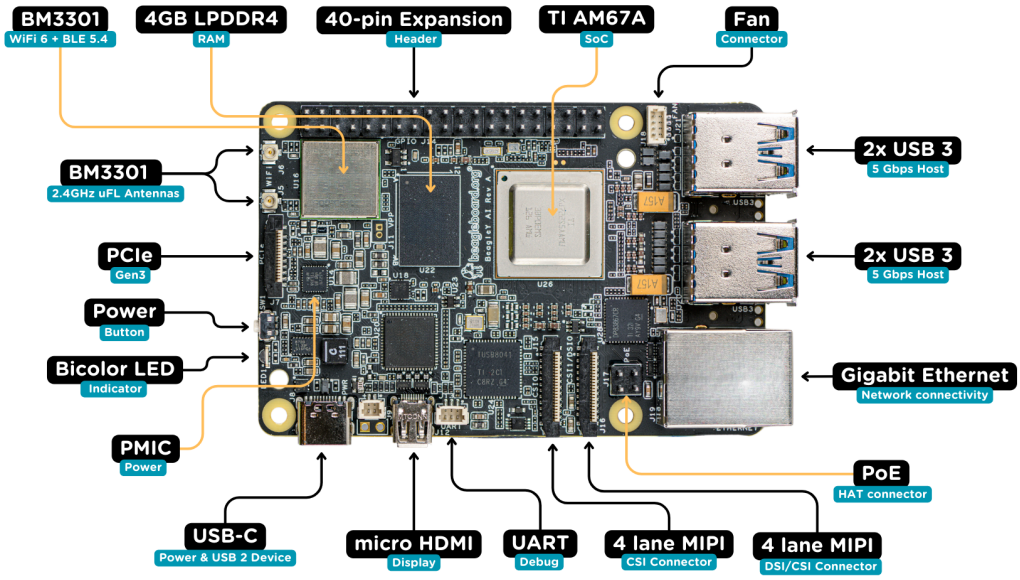


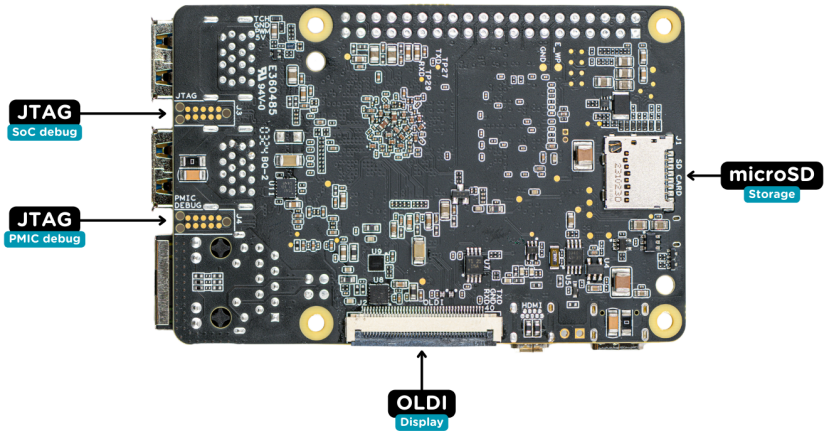
Table 1.2: BeagleY-AI board front components location

Feature	Description
WiFi/BLE	Beagleboard BM3301 with 802.11ax Wi-Fi & Bluetooth Low Energy 5.4 (BLE)
RAM	4GB LPDDR4
Expansion	40pin Expansion header compatible with HATs
SoC	TI AM67A Arm®Cortex®-A53 4 TOPS vision SoC with RGB-IR ISP for 4 cameras, machine vision, robotics, and smart HMI
Fan	4pin Fan connector
USB-A	4 x USB 3 TypeA ports supporting simultaneous 5Gbps operation host ports
Network Connectivity	Gigabit Ethernet
PoE	Power over Ethernet HAT connector
Camera/Display	1 x 4-lane MIPI camera/display transceivers, 1 x 4-lane MIPI camera
Debug UART	1 x 3-pin JST-SH 1.0mm debug UART port
Display Output	1 x HDMI display
USB-C	1 x Type-C port for power, and supports USB 2 device
PMIC	Power Management Integrated Circuit for 5V/5A DC power via USB-C with Power Delivery support
Bicolor LED	Indicator LED
Power button	ON/OFF button
PCIe	PCI-Express® Gen3 x 1 interface for fast peripherals (requires separate M.2 HAT or other adapter)

1.2.2 Back components

Table 1.3: BeagleY-AI board back components location

Feature	Description
Tag-Connect	1 x JTAG & 1 x Tag Connect for PMIC NVM Programming
Display output	1 x OLDI display
Storage	microSD card slot with support for high-speed SDR104 mode


JTAG
SoC debug
JTAG
PMIC debug
microSD
Storage
OLDI
Display

Chapter 2

BeagleY-AI Quick Start

2.1 What's included in the box?

When you purchase a BeagleY-AI, you'll get the following in the box:

1. BeagleY-AI with attached antenna.
2. Quick-start card

Todo: BeagleY-AI unboxing video

2.2 Getting started

To get started your BeagleY-AI you need the following:

1. 5V @ 3A power supply
2. MicroSD card (32GB)
3. *Boot Media (Software image)*

You may need additional accessories based on the mode of operation, you can use your BeagleY-AI in different ways.

1. *USB Tethering by directly connecting via USB type-c port*
2. *Headless connection via UART debug port*
3. *Standalone connection with Monitor and other peripherals attached*

Easiest option is to connect the board directly to your PC or Laptop using a USB type-C to type-c cable. There is only one USB type-C port on board, if you choose to use a dedicated power supply for first time setup, you may choose to access the board via any other methods listed above.

2.3 Power Supply

To power the board you can either connect it to a dedicated power supply like a mobile charger or a wall adapter that can provide 5V ≥ 3A. Checkout the docs power supply page for power supply recommendations.

Note: Instead of using a power supply or power adapter if you are using a Type-C to Type-C cable to connect the board to your laptop/PC then make sure it can supply at least 1000mA.

2.4 Boot Media (Software image)

We have two methods to prepare bootable microSD card, It is recommended to use *bb-imager*.

1. *bb-imager*
2. *Balena Etcher*

2.4.1 bb-imager

Download and install bb-imager for your operating system. Below are all the steps required to create a bootable microSD card with latest/recommended OS image for BeagleY-AI.

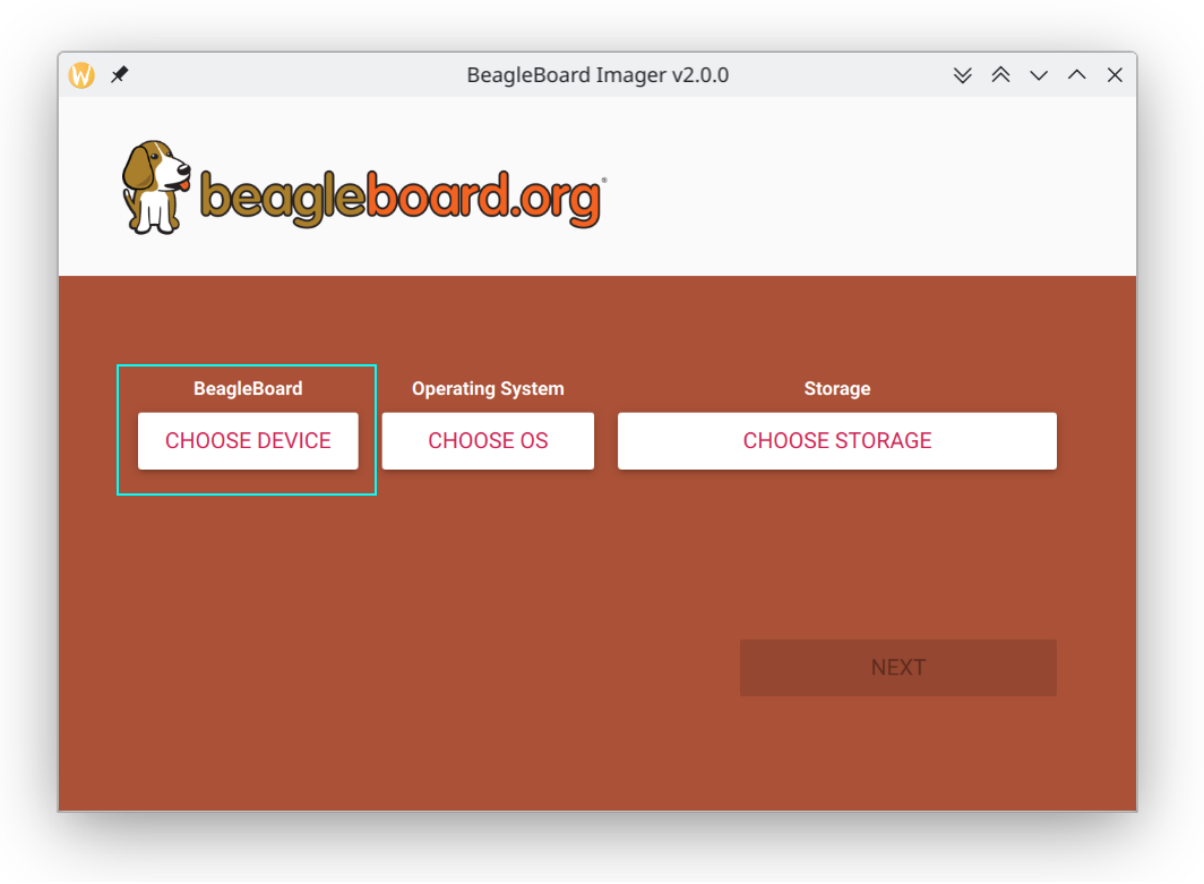


Fig. 2.1: Click on `CHOOSE DEVICE` button

2.4.2 Balena Etcher

Download and install Balena Etcher and then download the boot media from https://www.beagleboard.org/distros/beagley-ai-debian-12-5-2024-06-19-xfce. Flash it on a microSD card using Balena Etcher following the steps below:

1. Select downloaded boot media
2. Select microSD card
3. Flash!

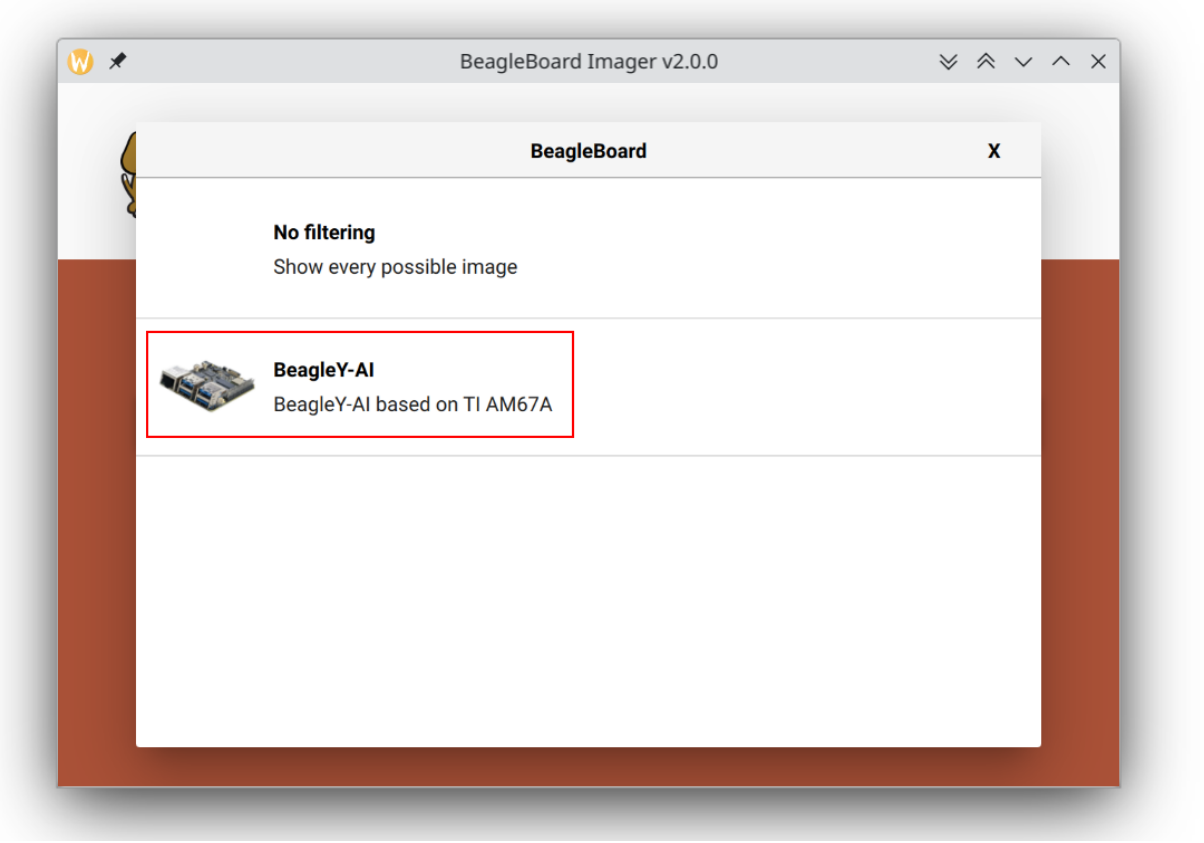


Fig. 2.2: Choose `BeagleY-AI` board

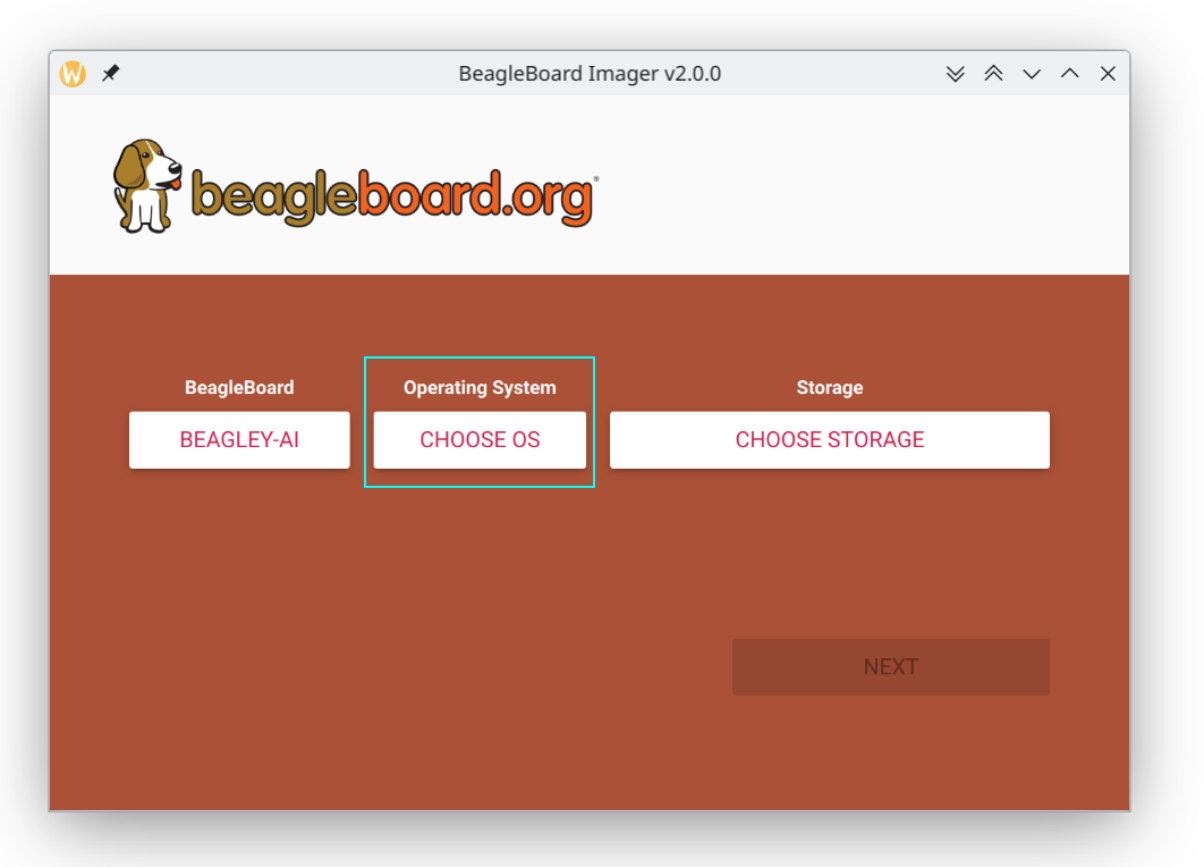


Fig. 2.3: Click on `CHOOSE OS` button

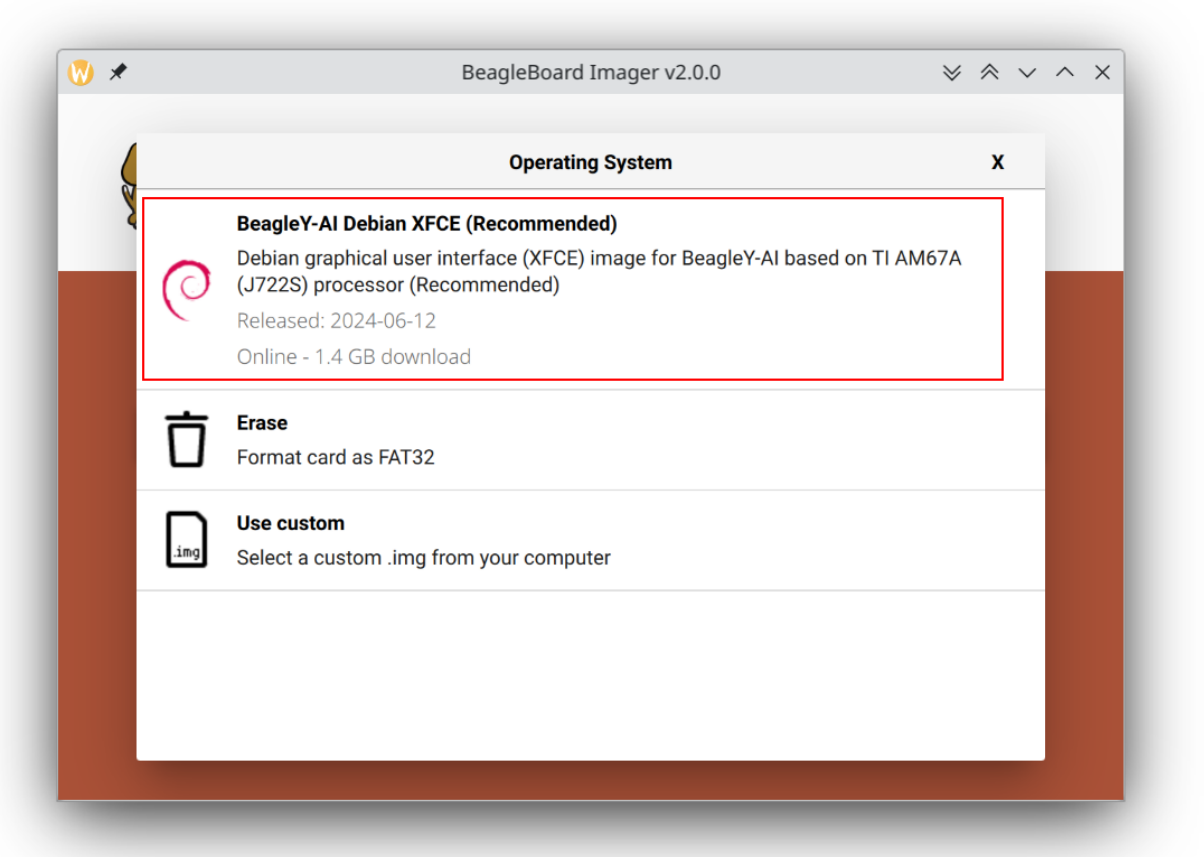


Fig. 2.4: Select `Recommended OS`

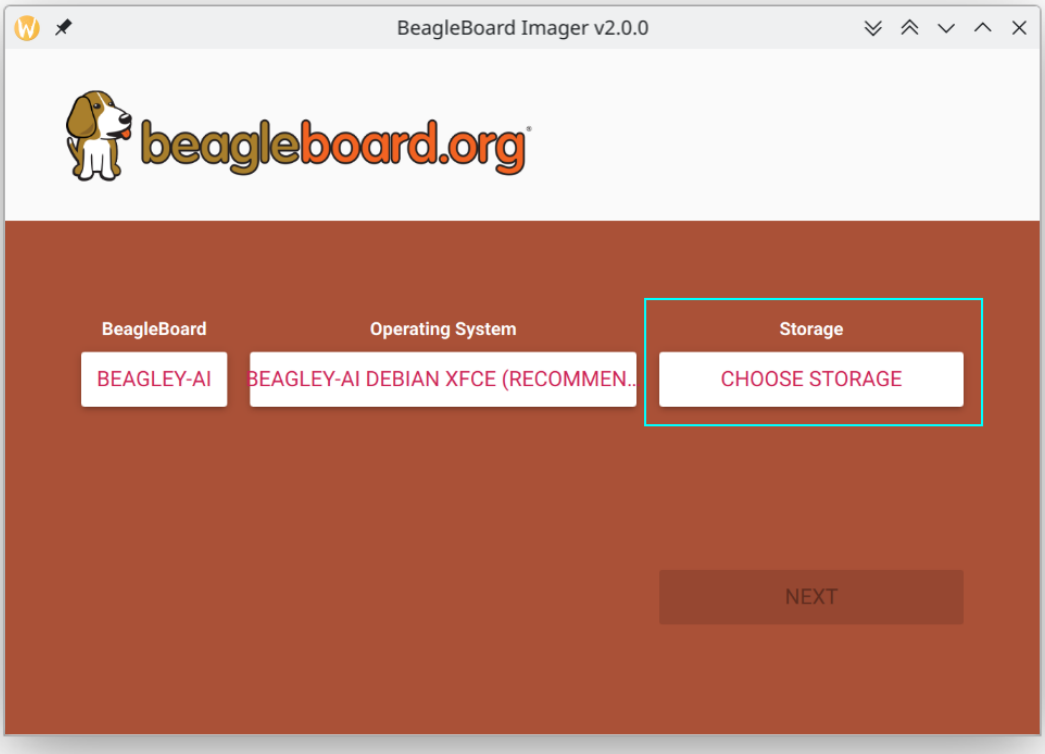


Fig. 2.5: Click on CHOOSE STORAGE button

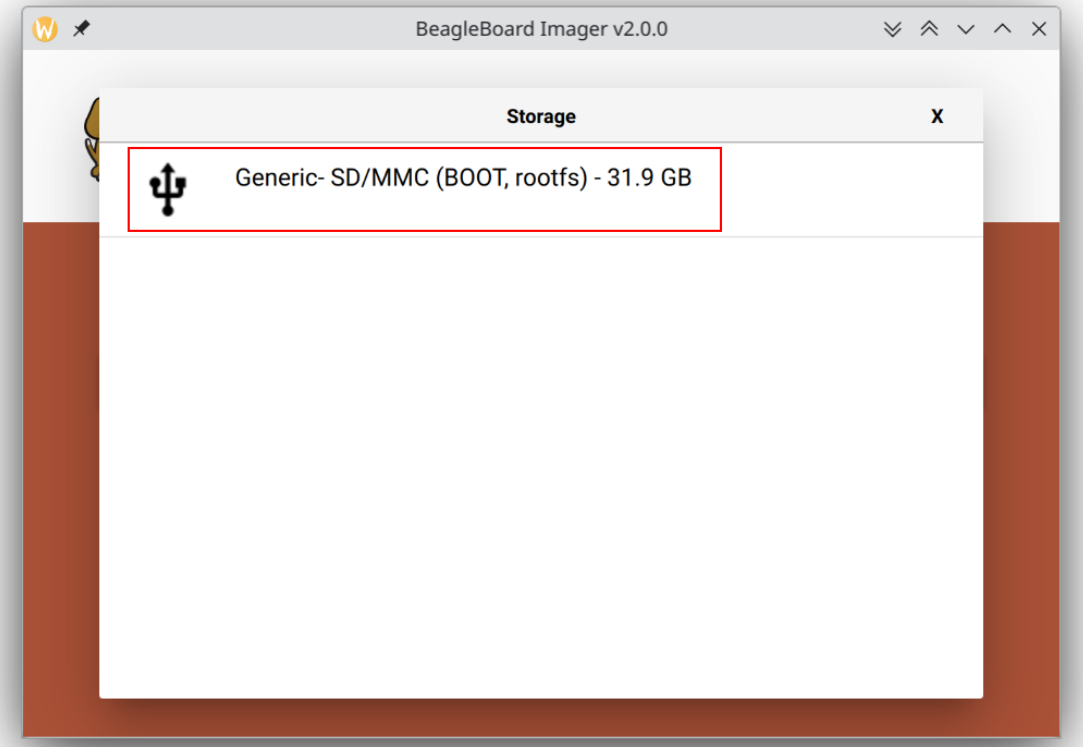


Fig. 2.6: Choose your microSD card

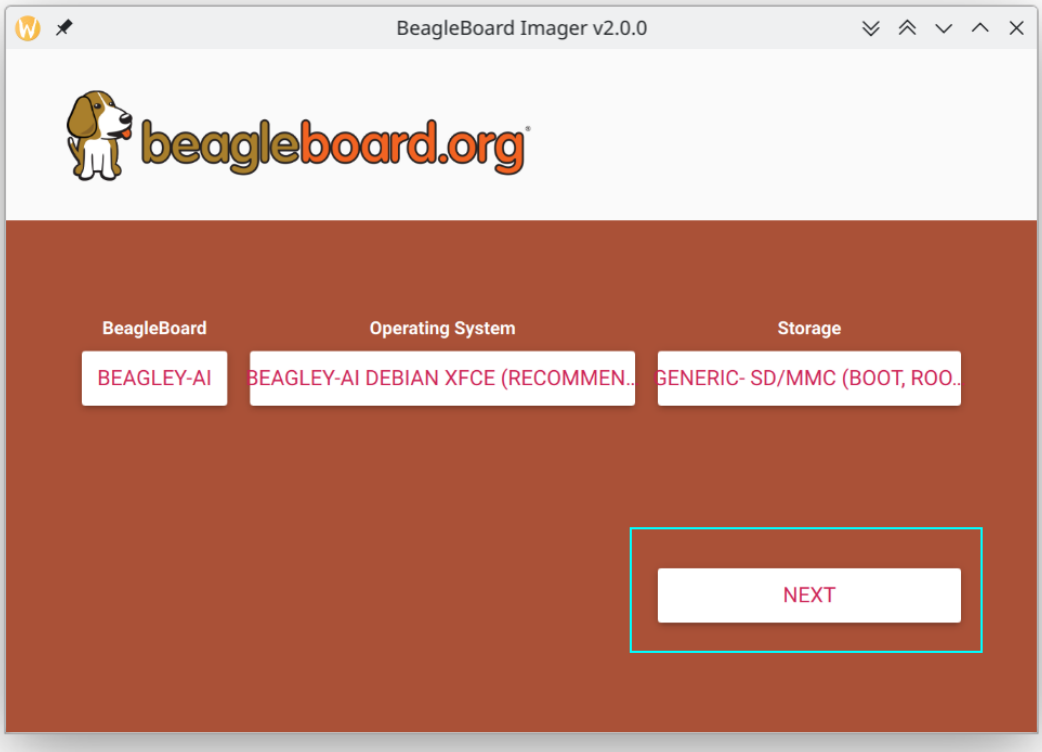


Fig. 2.7: Click on `Next` button

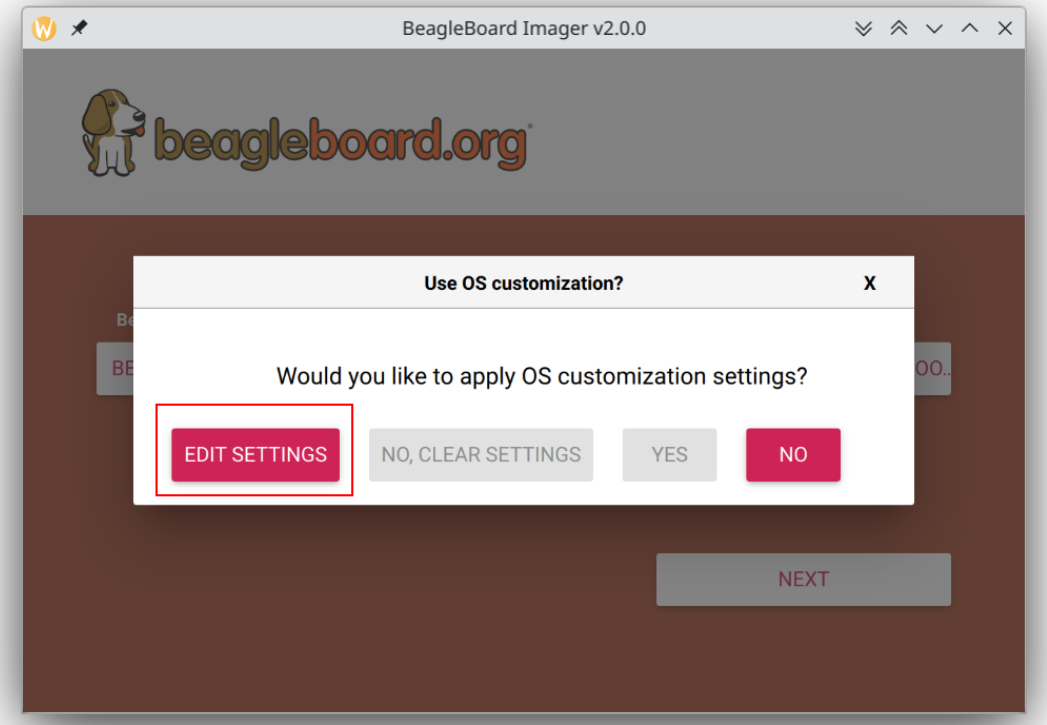


Fig. 2.8: Click on `EDIT SETTINGS` button

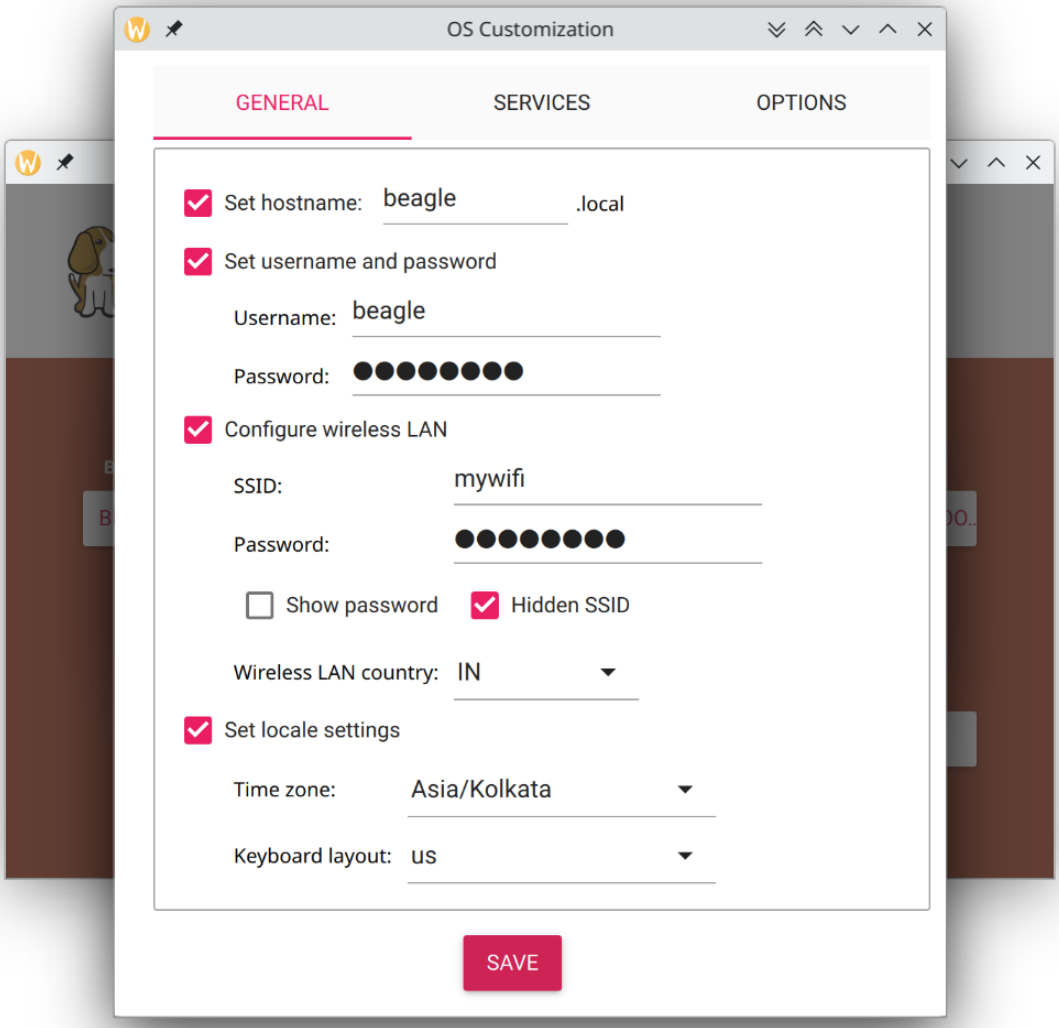


Fig. 2.9: Edit settings

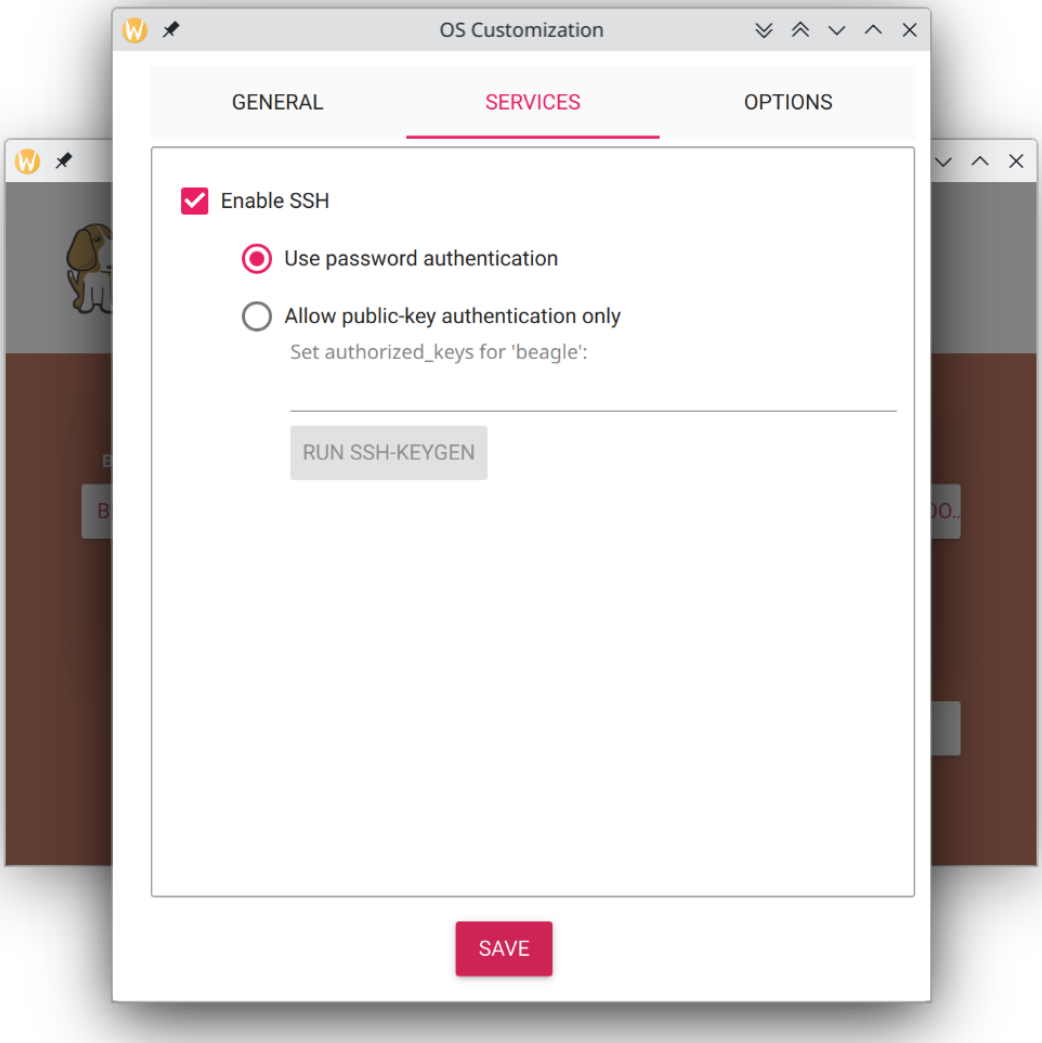


Fig. 2.10: Under SERVICES you can enable SSH

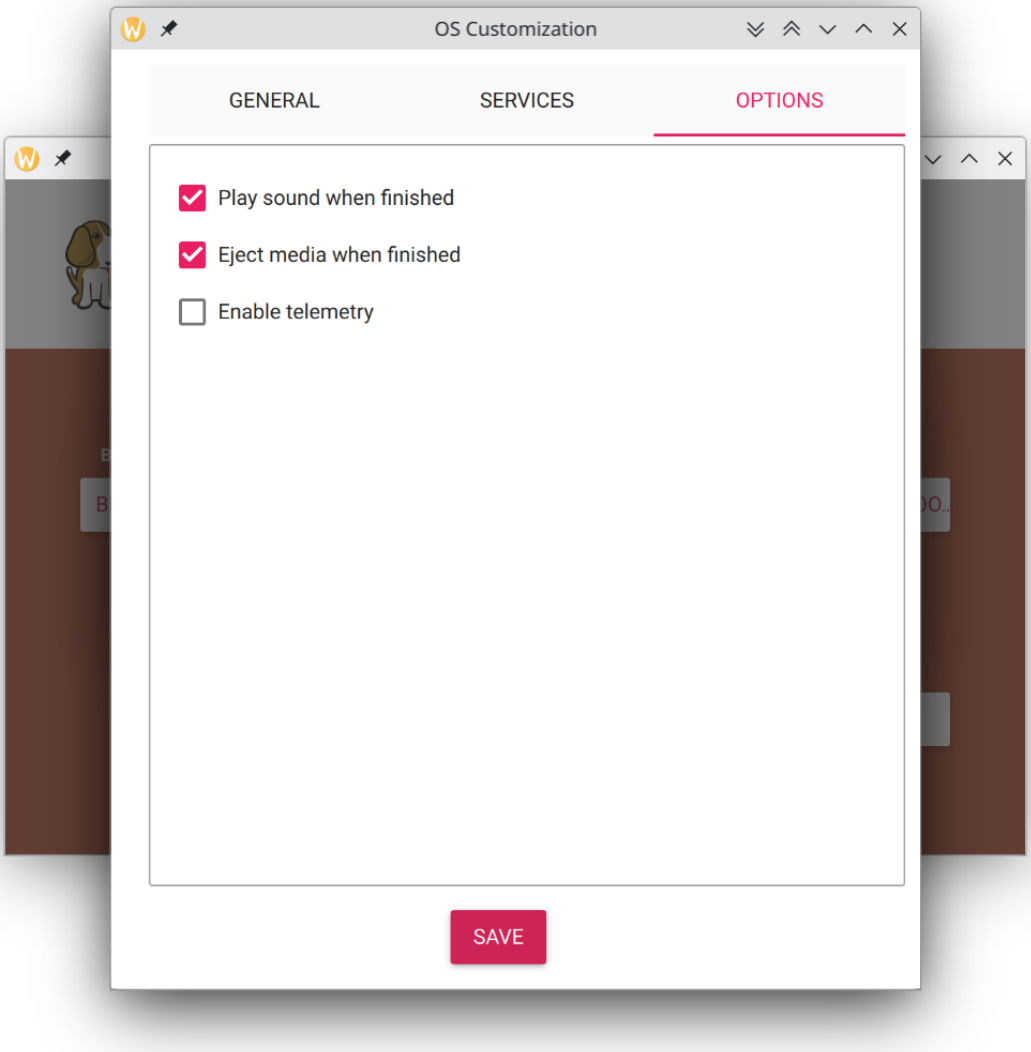


Fig. 2.11: Under OPTIONS you can enable to play sound when flashing is finished

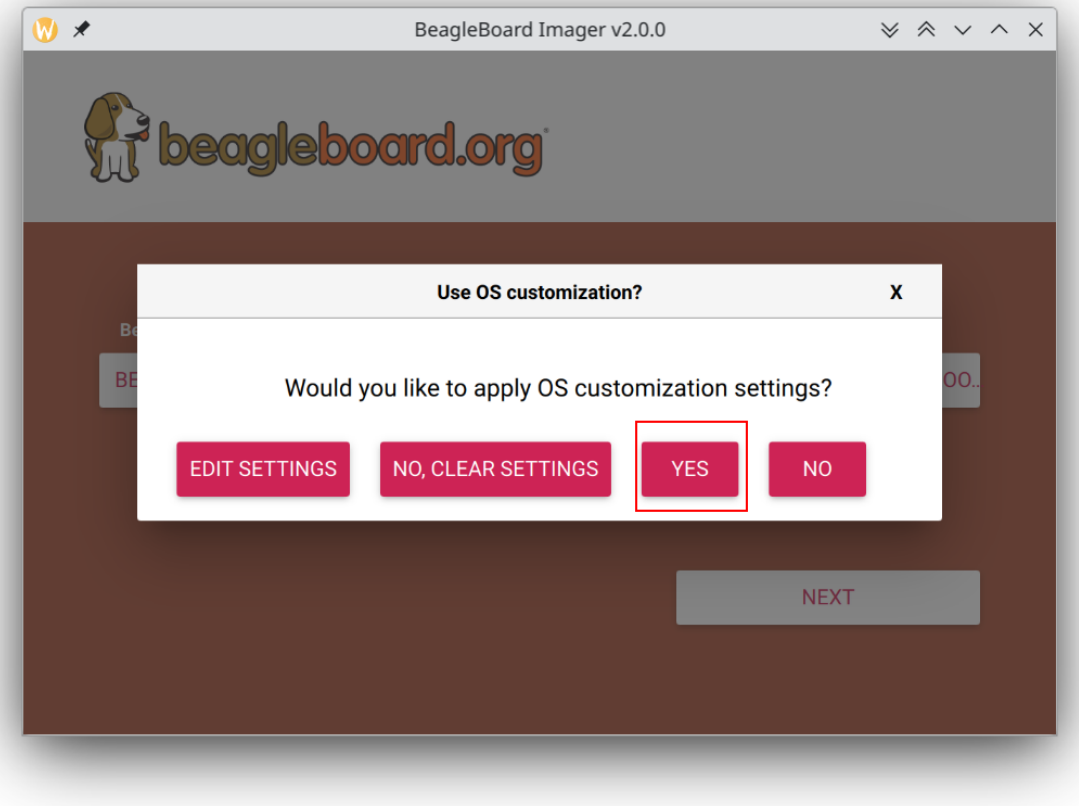


Fig. 2.12: Select YES to apply settings

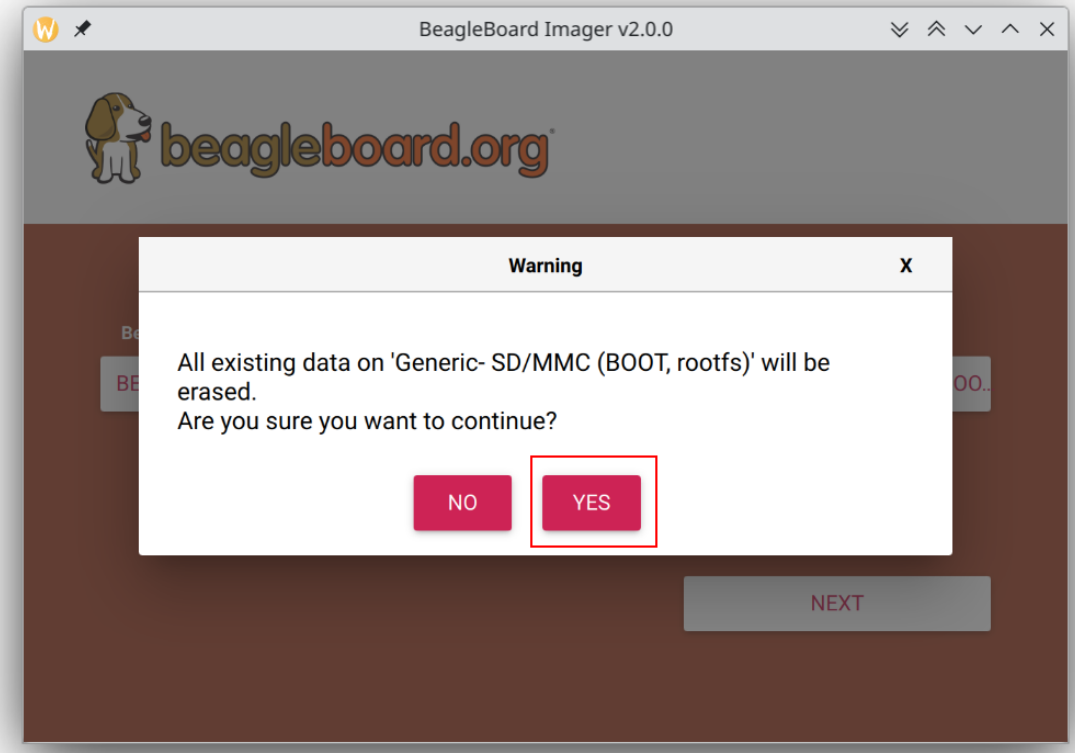


Fig. 2.13: Select YES again to confirm sdCard formatting

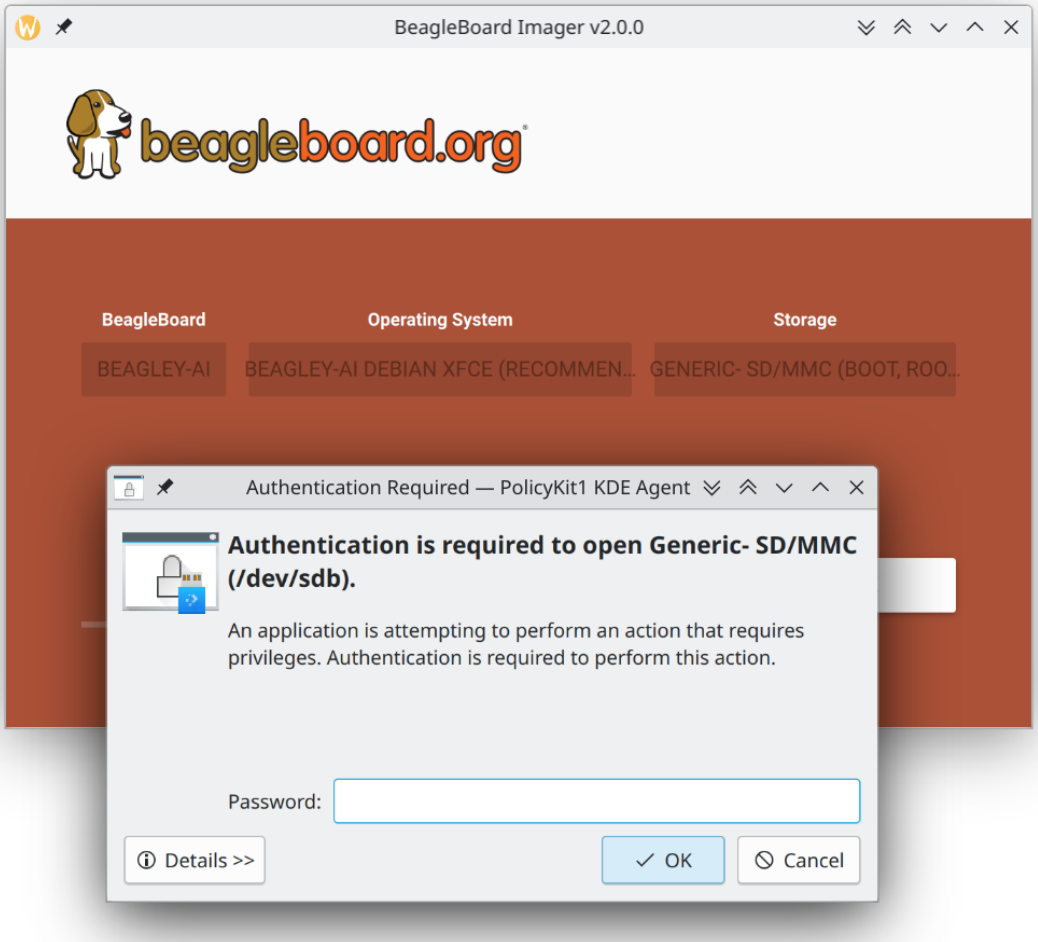


Fig. 2.14: Provide password to Authenticate the flashing process

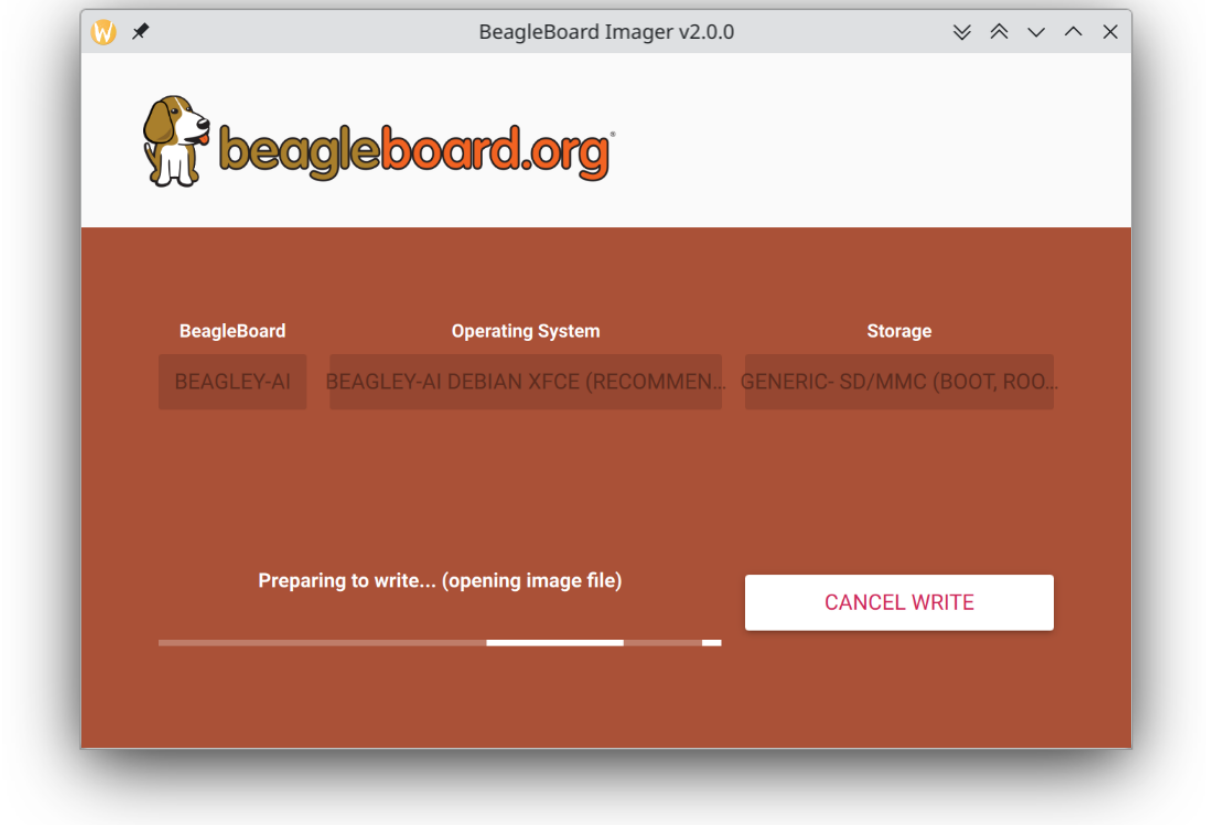


Fig. 2.15: Download image else automatically open cached image

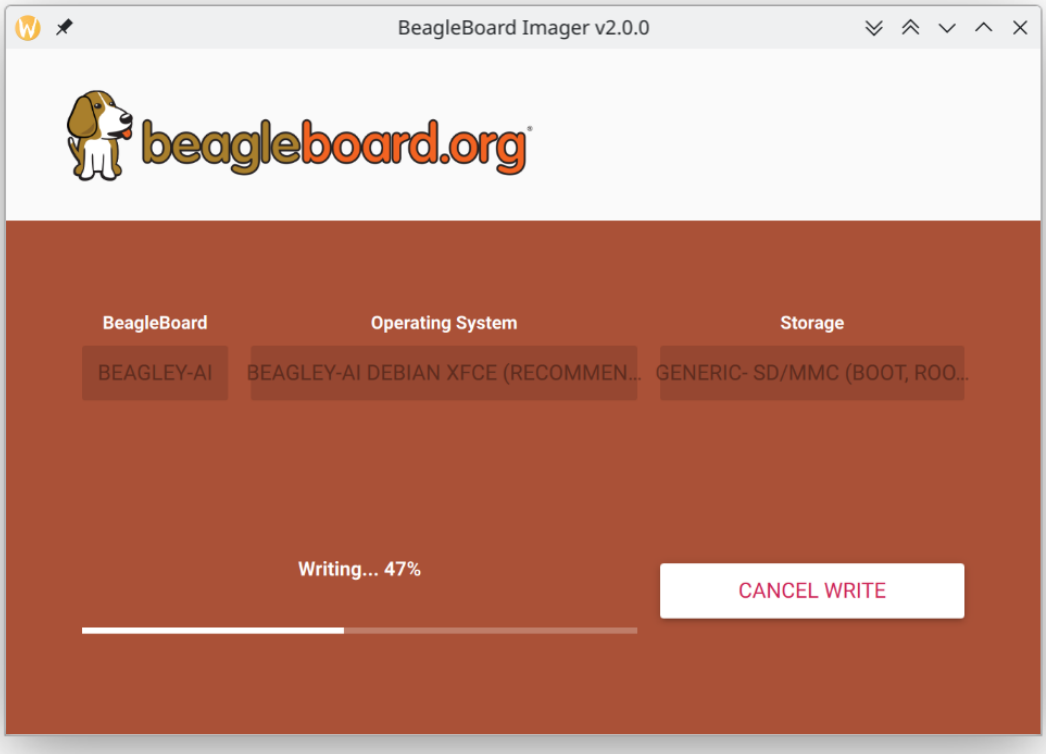


Fig. 2.16: Writing data to microSD card

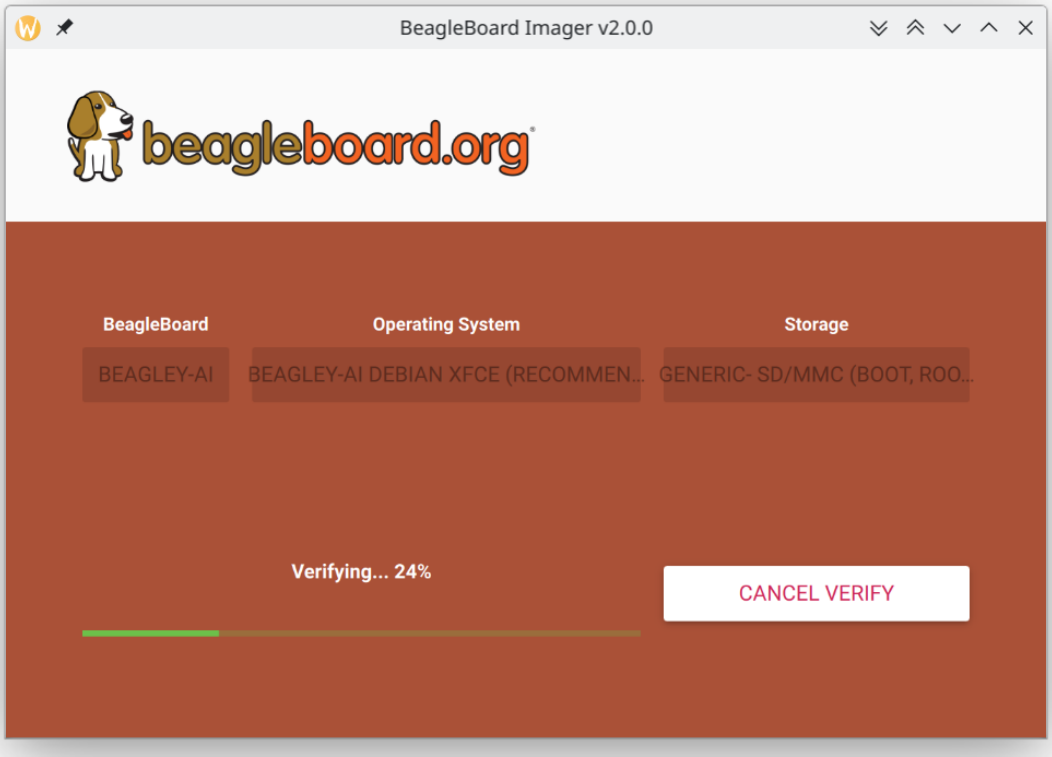


Fig. 2.17: Verifying flashed microSD card

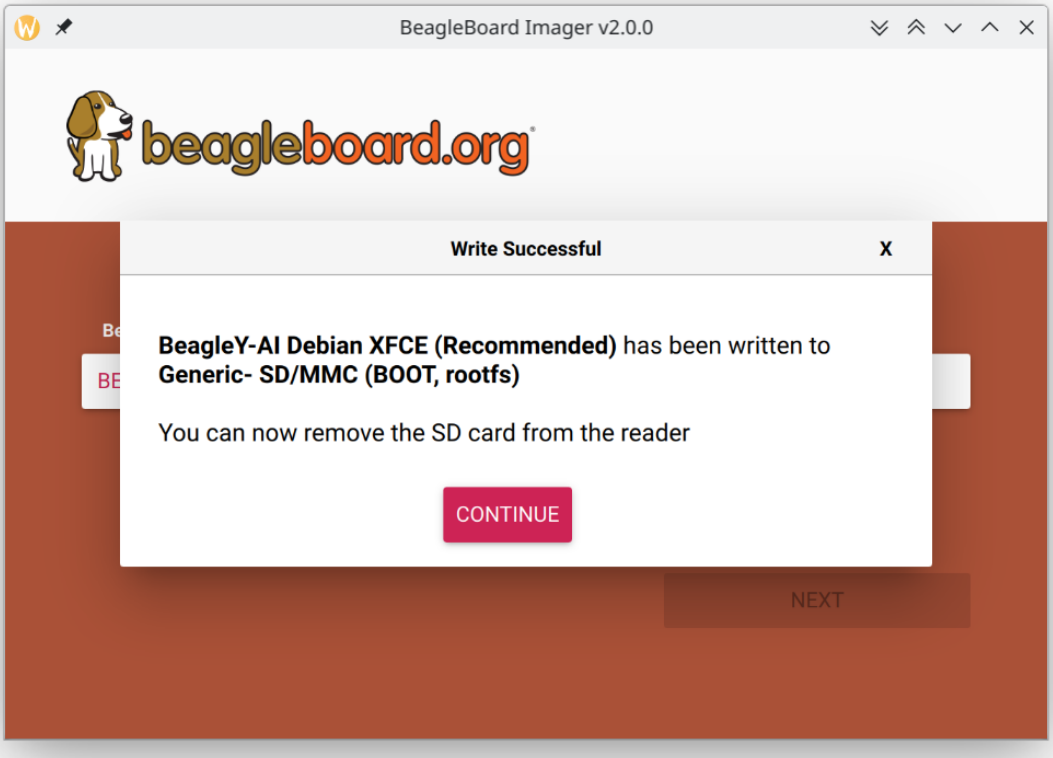


Fig. 2.18: microSD card is ready

Tip: For more detailed steps checkout the beagleboard-getting-started under support section of the documentation.

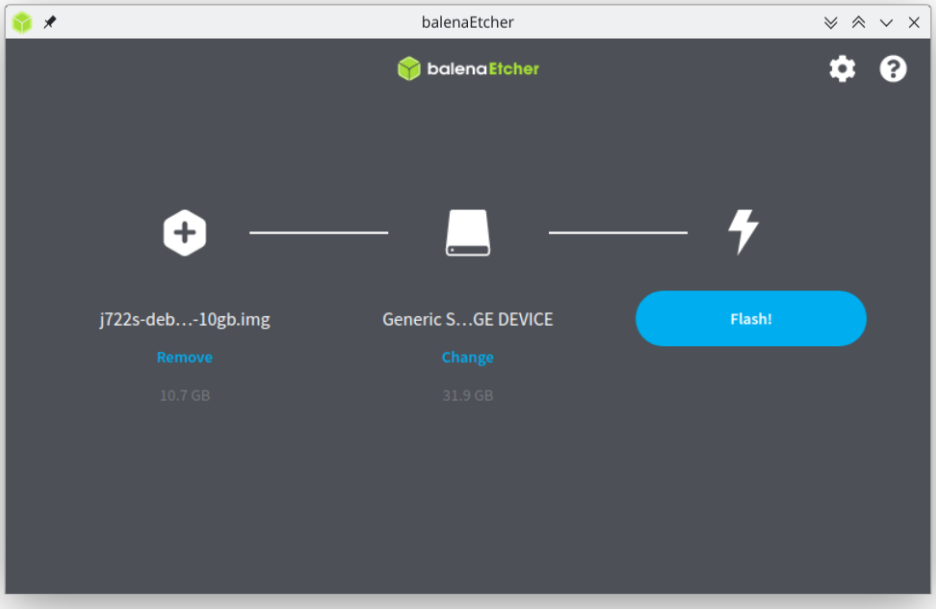


Fig. 2.19: Flashing BeagleY-AI boot image (software image) to microSD card

Once the microSD card is flashed you should see `BOOT` and `rootfs` mounted on your system as shown in image below,

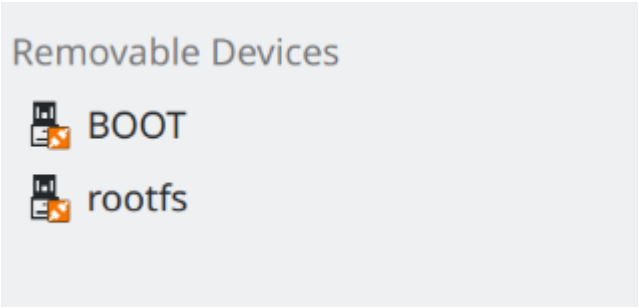


Fig. 2.20: Flashed microSD card mounted partitions

Under `BOOT` partition open `sysconf.txt` to edit login `username` and `password`.

In `sysconf.txt` file you have to edit the two lines highlighted below.

```
# user_name - Set a user name for the user (1000)
#user_name=beagle ①

# user_password - Set a password for user (1000)
#user_password=FooBar ②
```

① If `boris` is your username, update `#user_name=beagle` to `user_name=boris`

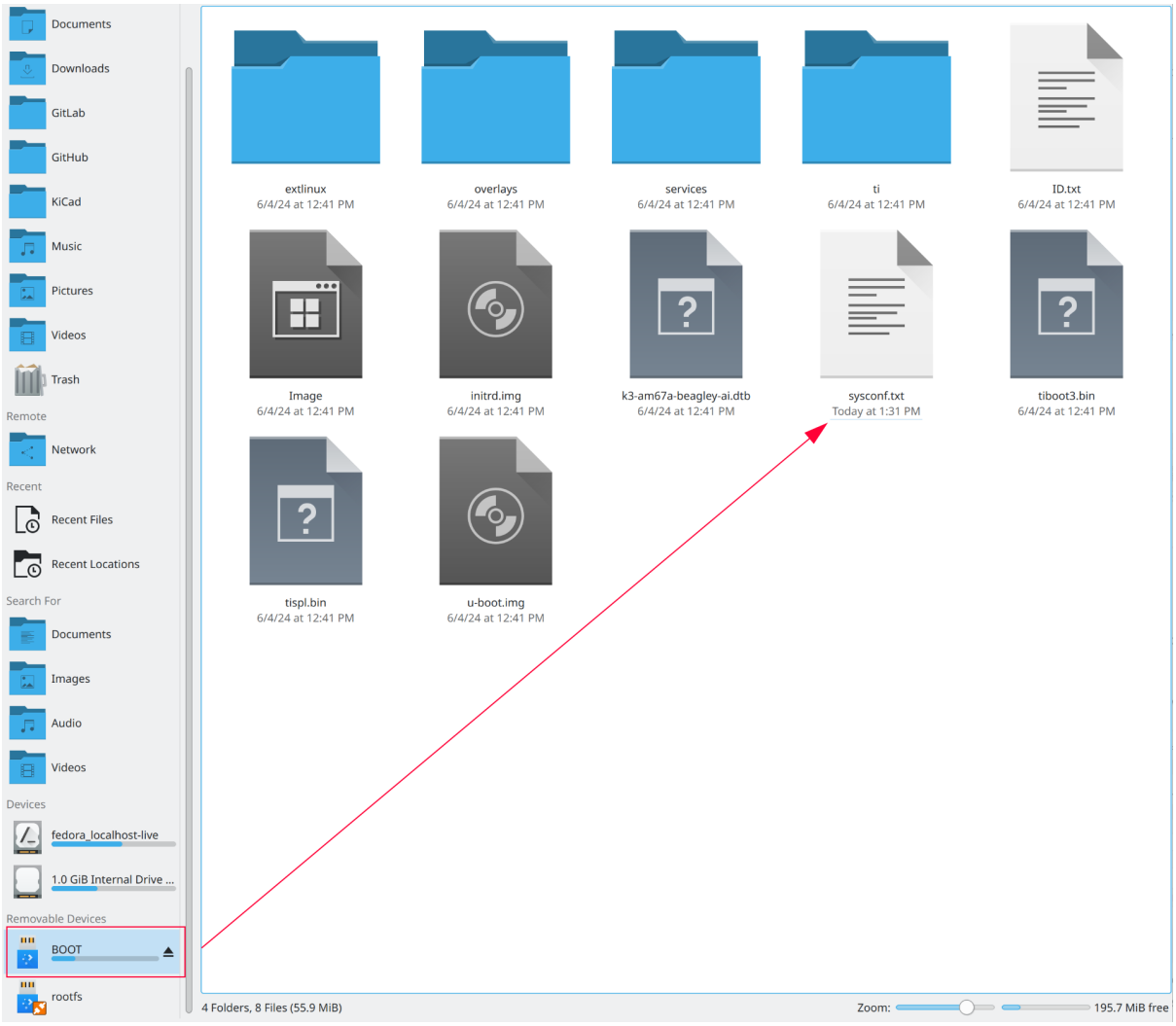


Fig. 2.21: sysconf file under BOOT partition

② If `bash` is your password, update `#user_password=FooBar` to `user_password=bash`

Important:

1. Make sure to remove `#` from `#user_name=` and `#user_password=` else the lines will be interpreted as a comment and your username & password will not be updated.
2. If you do not change your username and passord here then you will not see any output on your HDMI monitor when you do a *Standalone connection* setup.

Once username and password are updated, you can insert the microSD card into your BeagleY-AI as shown in the image below:

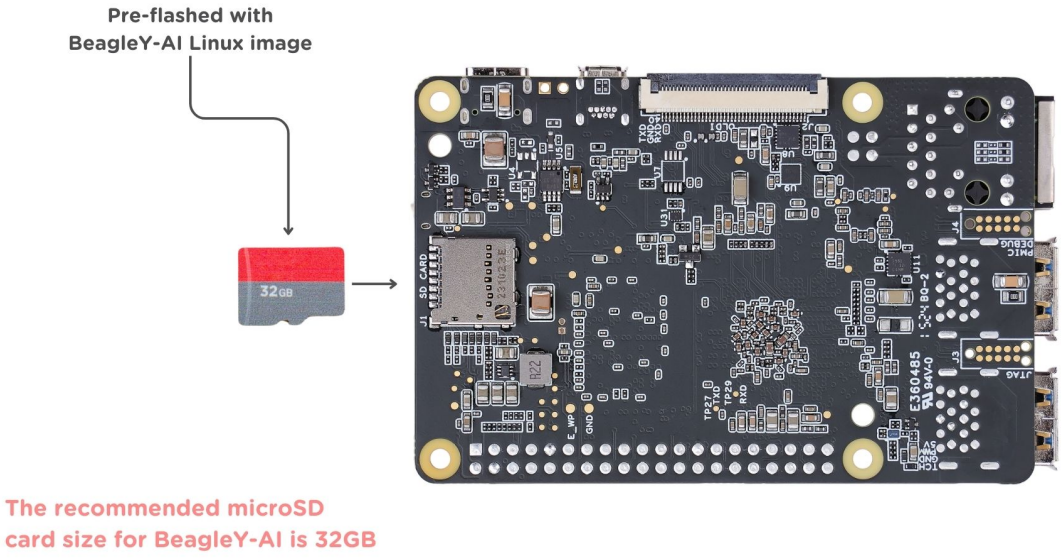


Fig. 2.22: Insert microSD card in BeagleY-AI

2.5 USB Tethering

Note: If you are using the board with a fan or running a computationally intensive task you should always power the board with a dedicated power supply that can supply 5V ≥ 3A (15W+).

As per USB standards these are the current at 5V that each downstream USB port type can (max) supply:

- USB Type-A 3.x port - 900mA (4.5W)
- USB Type-C 1.2 port - 1500mA (7.5W) to 3000mA (15W)

Thus it's recommended to use type-C to type-C cable.

To initially test your board, you can connect the board directly to your computer using a `type-C to type-C` cable shown in the image below.

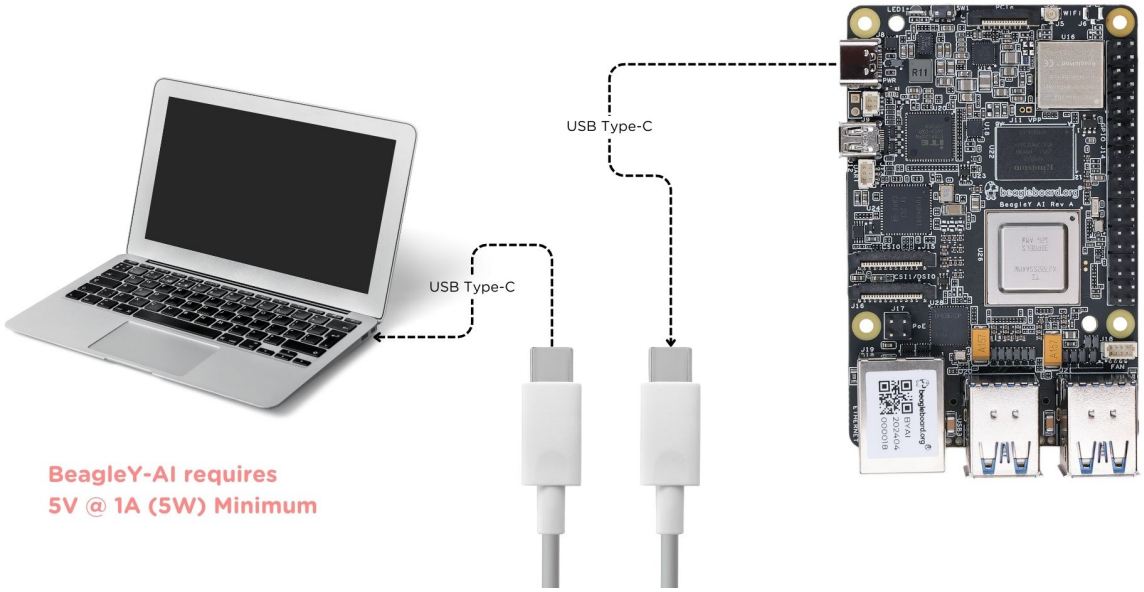


Fig. 2.23: BeagleY-AI tethered connection

2.5.1 SSH connection

After connecting, you should see the power LED glow, and soon just like with other Beagles, BeagleY-AI will create a virtual wired connection on your computer. To access the board, open up a terminal (Linux/Mac) or command prompt (Windows) and use the SSH command as shown below.

```
ssh debian@192.168.7.2
```

Important: Here `debian` is the default username, make sure to replace `debian` with the `username` you selected during *Boot Media (Software image)* preparation step.

Tip: If you are not able to find your beagle at `192.168.7.2`, checkout start-browse-to-beagle to resolve your connection issue.

Important: If you have not updated your default username and password during *Boot Media (Software image)*, you must update the default password at this step to something safer.

2.5.2 UART connection

Your BeagleY-AI board creates a UART connection (No additional hardware required) when tethered to a Laptop/PC which you can access using `Putty` of `tio`. On a linux machine it may come up as `dev/ttyACM*`, it will be different for Mac and Windows operating systems. To find serial port for your system you can checkout this guide.

- If you are on linux, try `tio` with default setting using command below,

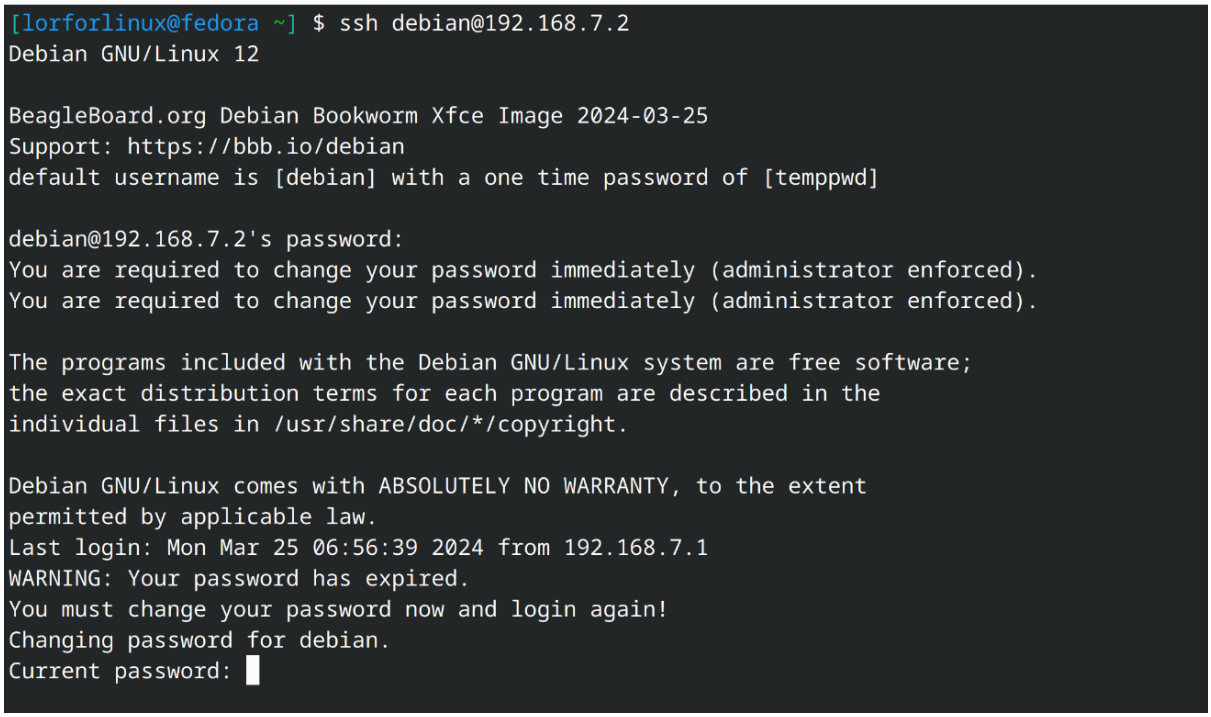


Fig. 2.24: BeagleY-AI SSH connection

```
tio /dev/ttyACM0
```

With this you have the access to BeagleY-AI terminal. Now, you can connect your board to *WiFi*, try out all the *cool demos* and explore all the other ways to access your BeagleY-AI listed below.

- *Connecting to WiFi*
- *Demos and tutorials*

2.5.3 Headless connection

If you want to run your BeagleY-AI in headless mode, you need Raspberry Pi Debug Probe or similar serial (USB to UART) adapter. Connect your UART debug probe to BeagleY-AI as shown in the image below. After making the connection you can use command line utility like `tio` on Linux of Putty on any operating system. Check *UART connection* for more information.

2.5.4 Standalone connection

Important: Make sure to update your `username` and `password` during *Boot Media (Software image)* preparation step else you'll not see any output on you HDMI monitor.

To setup your BeagleY-AI for standalone usage, you need the following additional accessories,

1. HDMI monitor
2. micro HDMI to full-size HDMI cable
3. Wireless keyboard & mice combo
4. Ethernet cable (Optional)

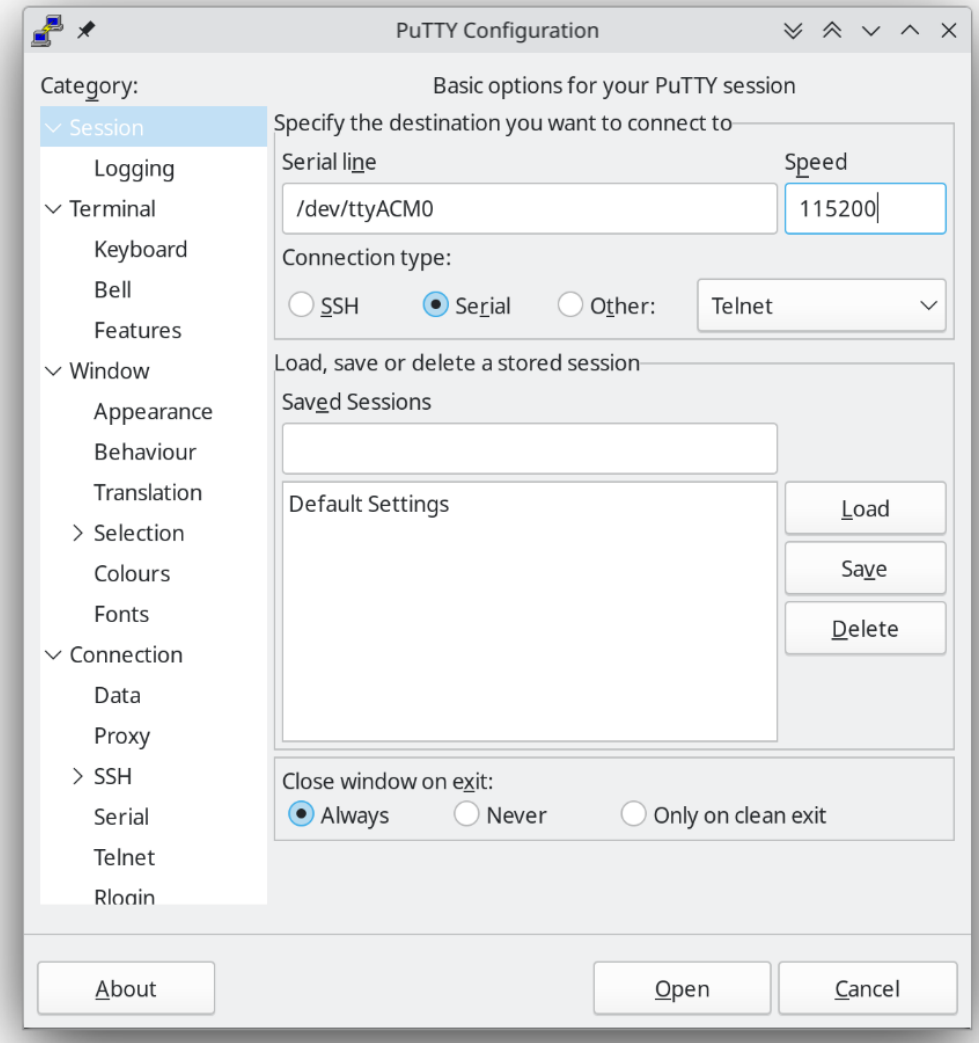


Fig. 2.25: Putty serial connection

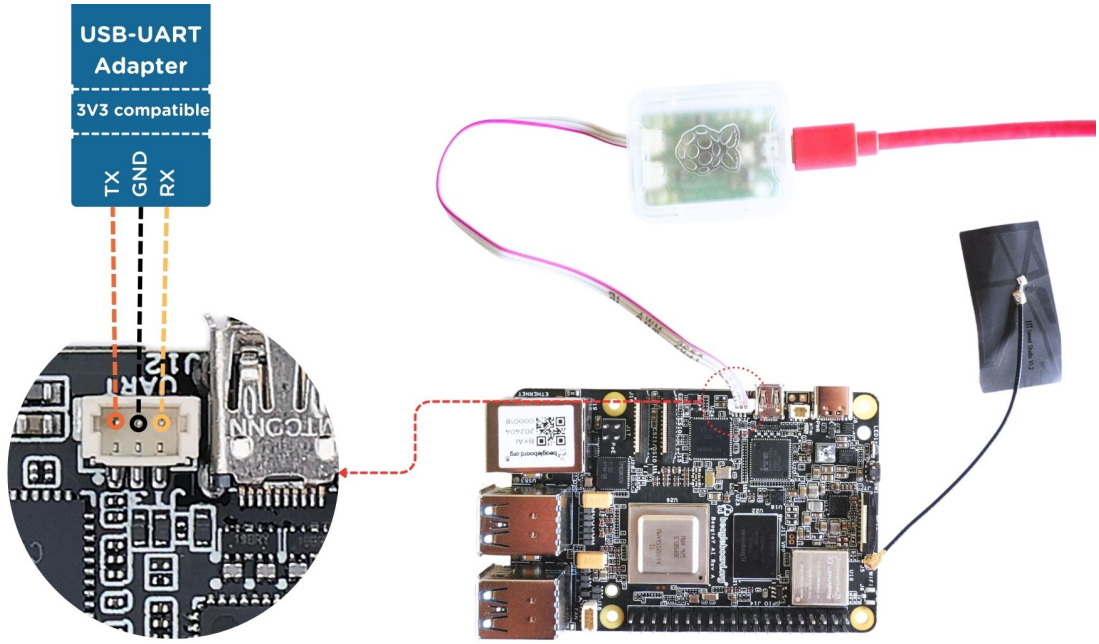


Fig. 2.26: Connecting Raspberry Pi debug probe to BeagleY-AI

Make sure you have the microSD card with boot media (software image) inserted in to the BeagleY-AI. Now connect,

1. microHDMI to BeagleY-AI and full size HDMI to monitor
2. keyboard and mice combo to one of the four USB port of BeagleY-AI
3. Power supply to USB type-c connector of BeagleY-AI

The connection diagram below provides a clear representation of all the connections,

If everything is connected properly you should see four penguins on your monitor.

When prompted, login using the credentials you updated during *Boot Media (Software image)* preparation step.

Important: You can not update login credentials at this step, you must update them during boot media (software image) micrSD card flashing or USB tethering step!

Once logged in you should see the splash screen shown in the image below:

Test network connection by running `ping 8.8.8.8`

Explore and build with your new BeagleY-AI board!

2.6 Connecting to WiFi

The onboard `BM3301` can connect to any 2.5GHz wifi access point. We have two options to connect to WiFi,

1. *nmtui*
2. *iwctl*

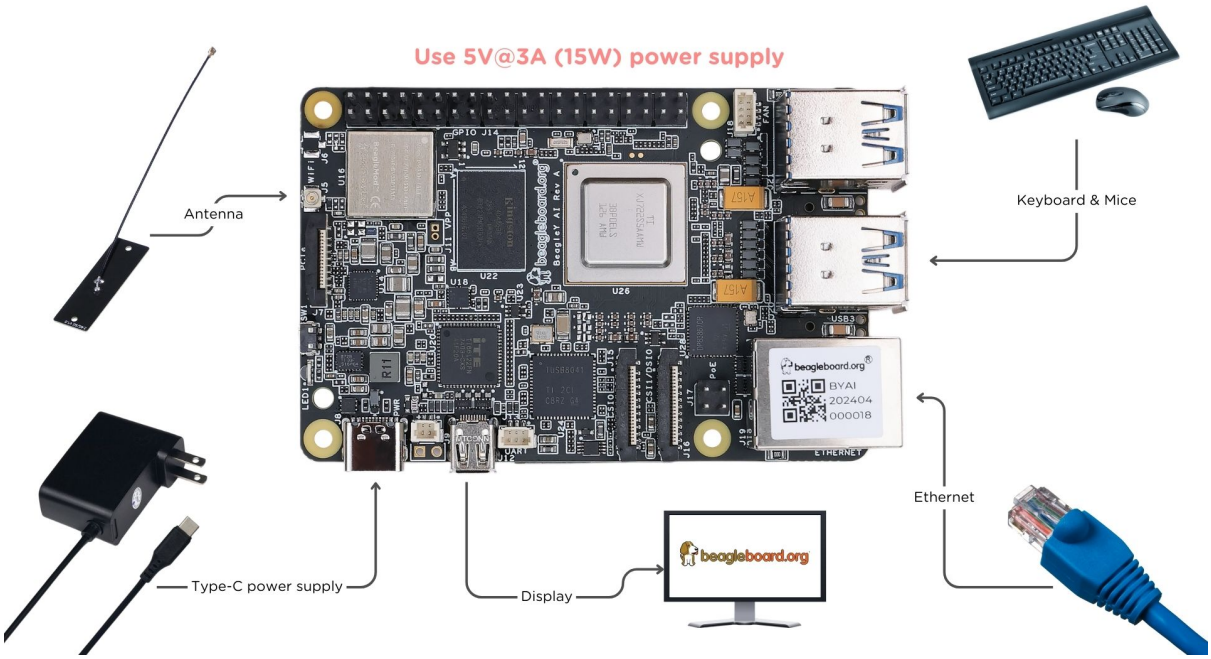


Fig. 2.27: BeagleY-AI standalone connection

Fig. 2.28: BeagleY-AI boot penguins

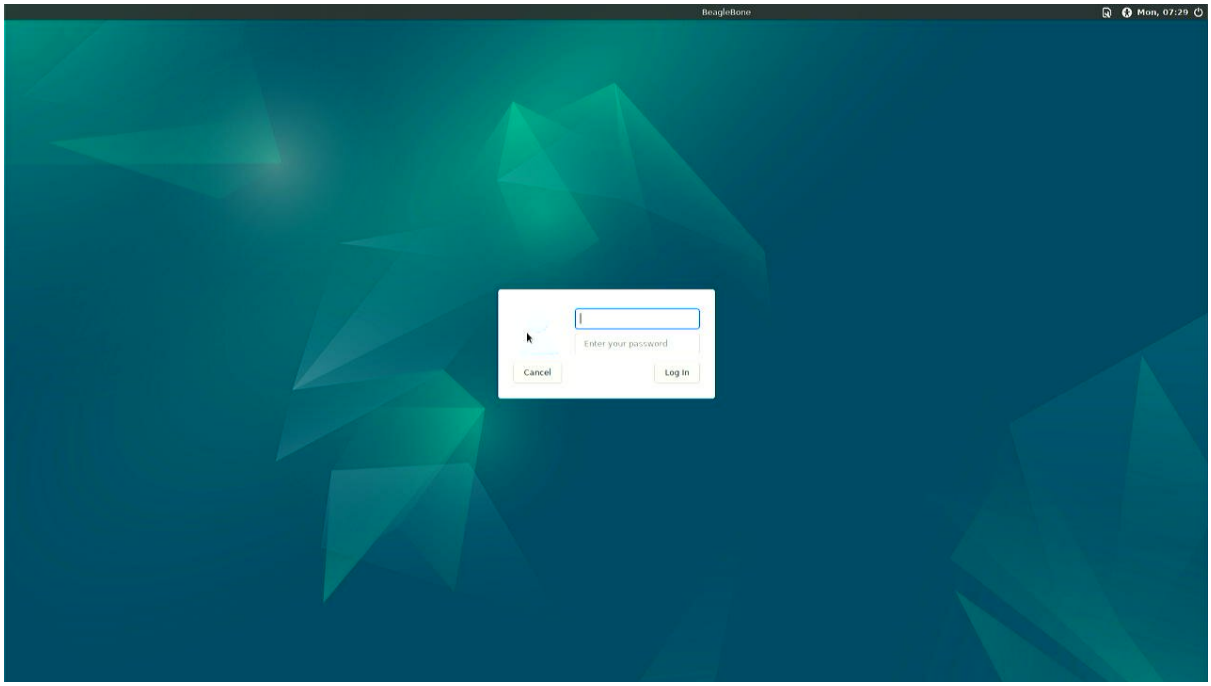


Fig. 2.29: BeagleY-AI XFCE desktop login

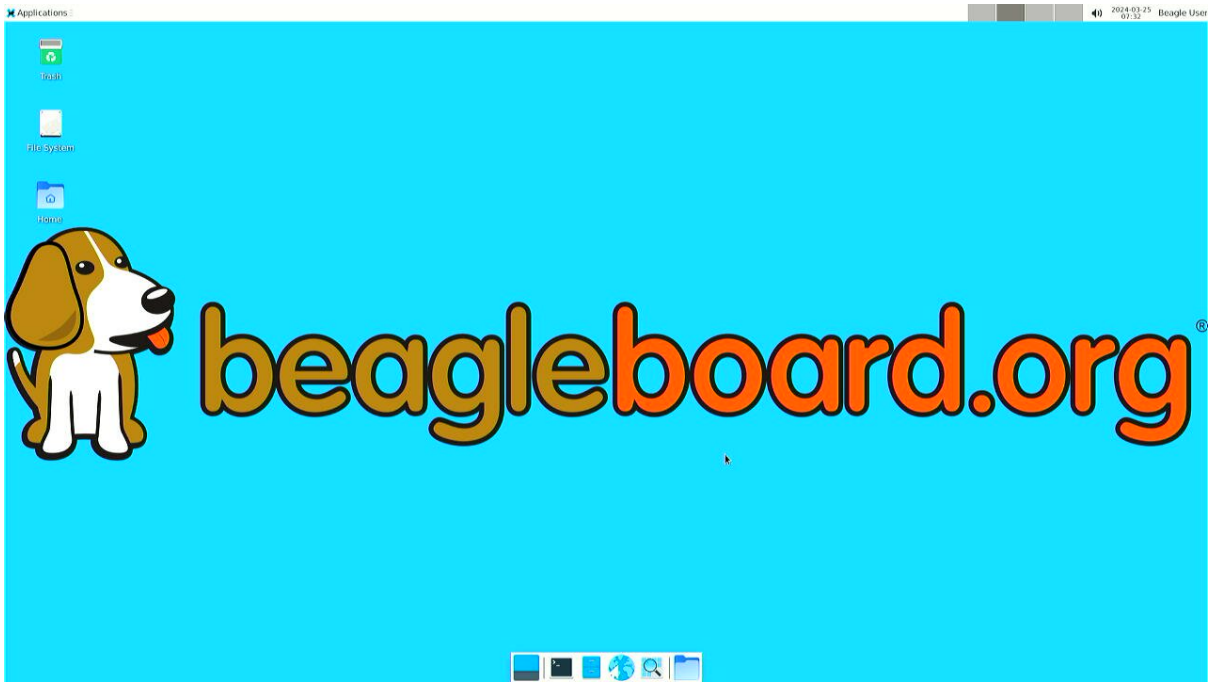


Fig. 2.30: BeagleY-AI XFCE home screen

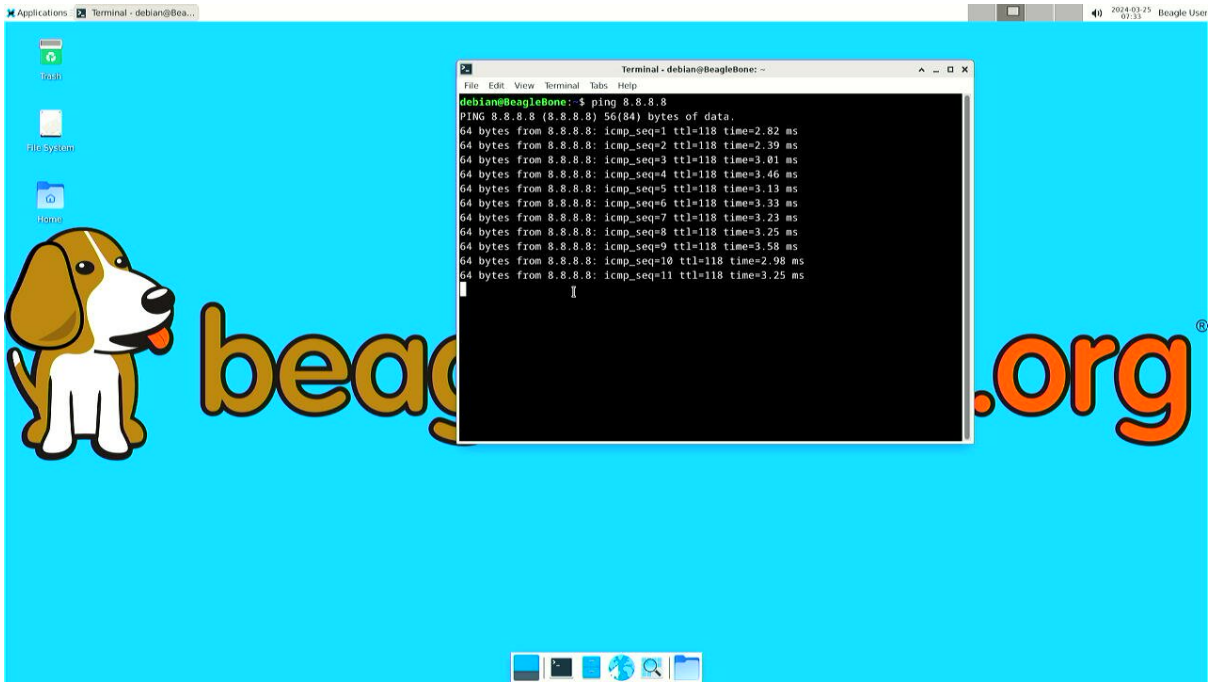


Fig. 2.31: BeagleY-AI network ping test

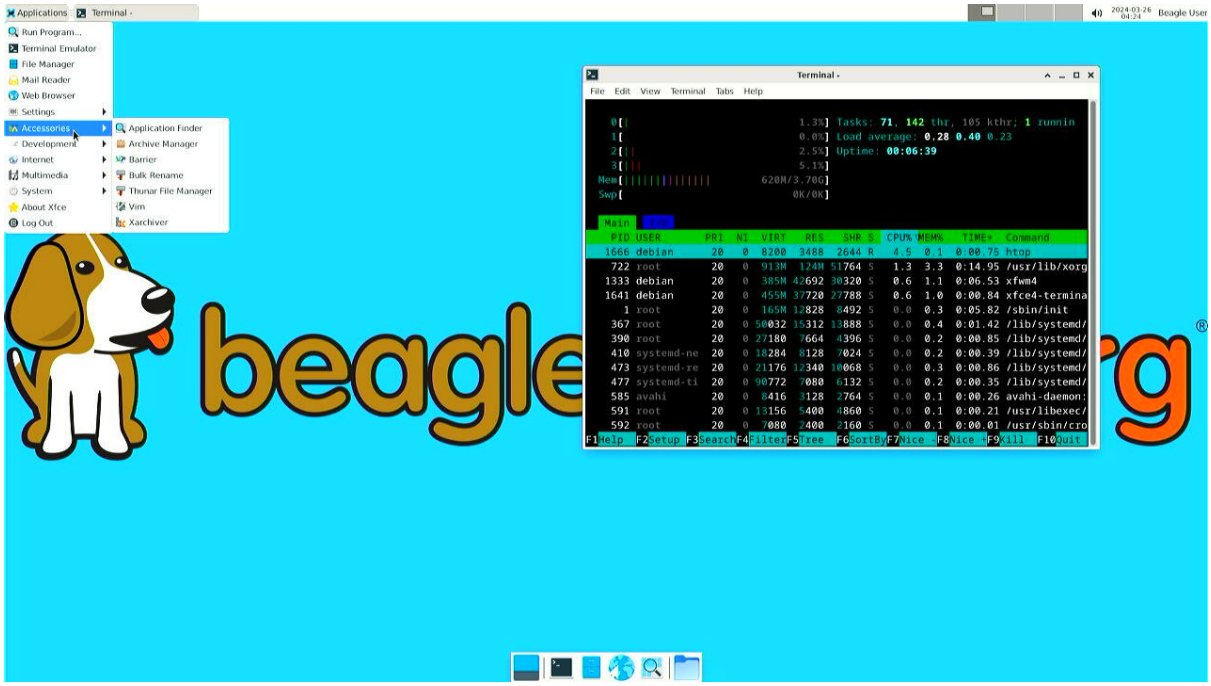


Fig. 2.32: BeagleY-AI running htop

2.6.1 nmtui

- Enable `NetworkManager`

```
sudo systemctl enable NetworkManager
```

- Start `NetworkManager`

```
sudo systemctl start NetworkManager
```

- Start `nmtui` application

```
sudo nmtui
```

- To navigate, use the `arrow keys` or press `Tab` to step forwards and press `Shift+Tab` to step back through the options. Press `Enter` to select an option. The `Space bar` toggles the status of a check box.
- You should see a screen as shown below, here you have to press `Enter` on `Acticate a connection` option to activate wired and wireless connection options.

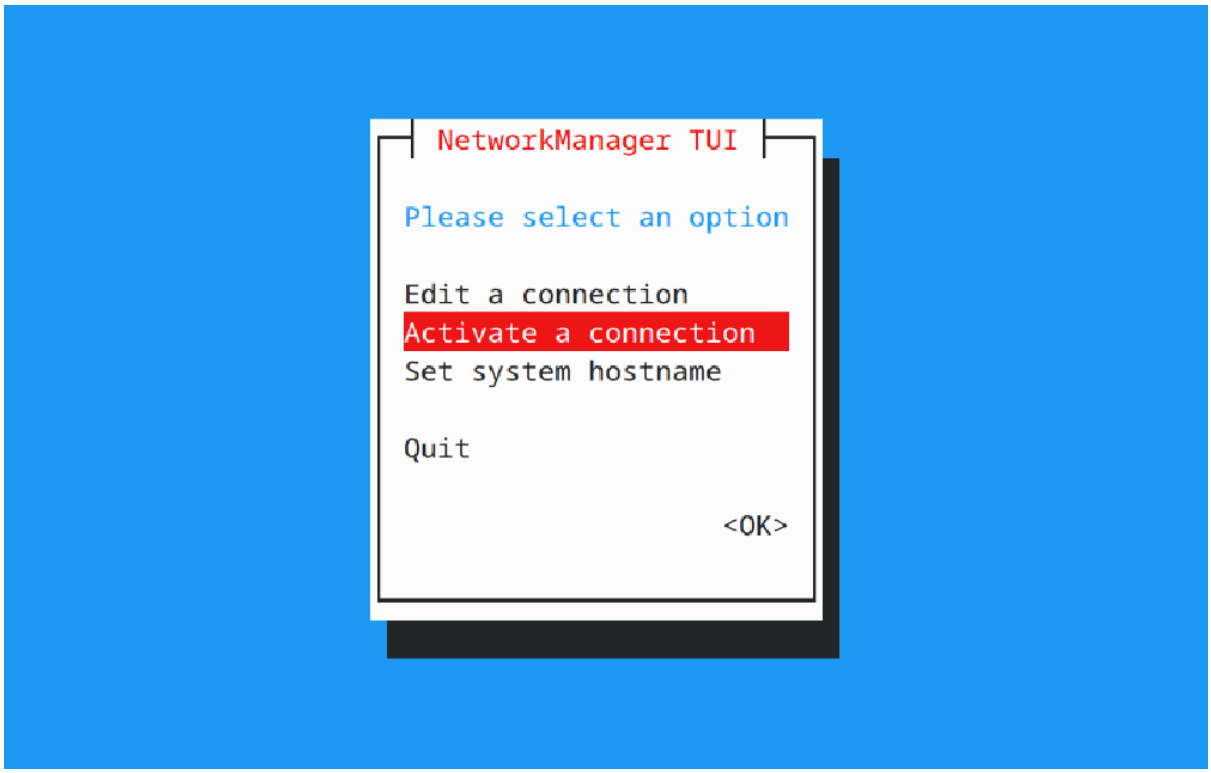


Fig. 2.33: NetworkManager TUI

There under `WiFi` section press `Enter` on desired access point and provide password to connect. When successfully connected press `Esc` to get out of the `nmtui` application window.

2.6.2 iwctl

Once board is fully booted and you have access to the shell, follow the commands below to connect to any WiFi access point,

- To list the wireless devices attached, (you should see wlan0 listed)

```
iwctl device list
```

- Scan WiFi using,

```
iwctl station wlan0 scan
```

- Get networks using,

```
iwctl station wlan0 get-networks
```

- Connect to your wifi network using,

```
iwctl --passphrase "<wifi-pass>" station wlan0 connect "<wifi-name>"
```

- Check wlan0 status with,

```
iwctl station wlan0 show
```

- To list the networks with connected WiFi marked you can again use,

```
iwctl station wlan0 get-networks
```

- Test connection with ping command,

```
ping 8.8.8.8
```

2.7 Attach cooling fan

To attached the Raspberry Pi cooling fan to BeagleY-AI you have to follow these steps,

1. Clean the surface of BeagleY-AI with a microfiber cloth or electronics safe cleaning brush.
2. Gently pull the pre-cut (blue) thermal pads from cooling fan surface and transfer them to the most heating parts of BegleY-AI like CPU and RAM.
3. Connect the fan cable, then carefully place the flat part of cooling fan on BeagleY-AI. Now, gently apply force on spring loaded push pins to securely attach the cooling fan.

2.8 Demos and Tutorials

- *Booting from NVMe Drives*

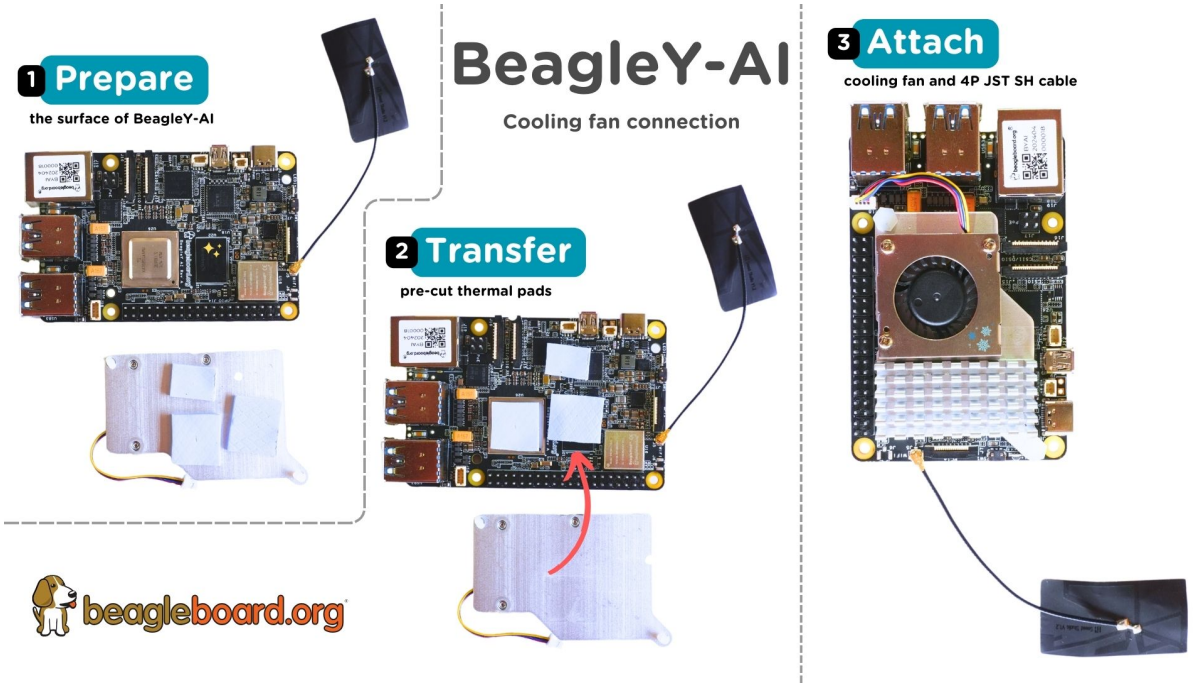


Fig. 2.34: Attaching cooling fan to BeagleY-AI

Chapter 3

Design and Specifications

Todo: Add details about all the schematic sections.

If you want to know how BeagleY-AI is designed and the detailed specifications, then this chapter is for you. We are going to attempt to provide you a short and crisp overview followed by discussing each hardware design element in detail.

Tip: For board files, 3D model, and more, you can checkout the BeagleY-AI repository on OpenBeagle.

3.1 Block Diagram and Overview

3.2 Processor

The AM67A processor from Texas Instruments is a highly integrated SoC with an Automotive pedigree. It may be referenced by TI documentation by it's superset J722s/TDA4AEN.

It's primary compute cluster revolves around 4xARM Cortex-A53 Cores running at 1.4Ghz.

An MCU subsystem consisting of an ARM Cortex-R5F running at up to 800Mhz is also available for user applications and is especially useful for real-time IO applications.

For very advanced users, two additional R5 cores are also present, but they are normally reserved for Device and Run-time Management of the SoC typically.

2x C7x DSPs with MMA support are intended for use as Deep Learning Accelerators for things like AI Vision, with up to 2TOPS each.

An Imagination BXS-4-64 GPU rounds out the compute cluster, with a dedicated video encoder/decoder available for multimedia tasks.

The SoC features advanced high speed connectivity, including USB3.1, PCIe and more.

Secure Boot is also available with the ability burn One-Time-Programmable (OTP) eFUSES by energizing the VPP test pads.

3.3 Boot Modes

The default boot mode for BeagleY-AI is the SD Card Interface. If no SD card is present, the BootROM on the AM67A SoC is going to try booting from Ethernet.

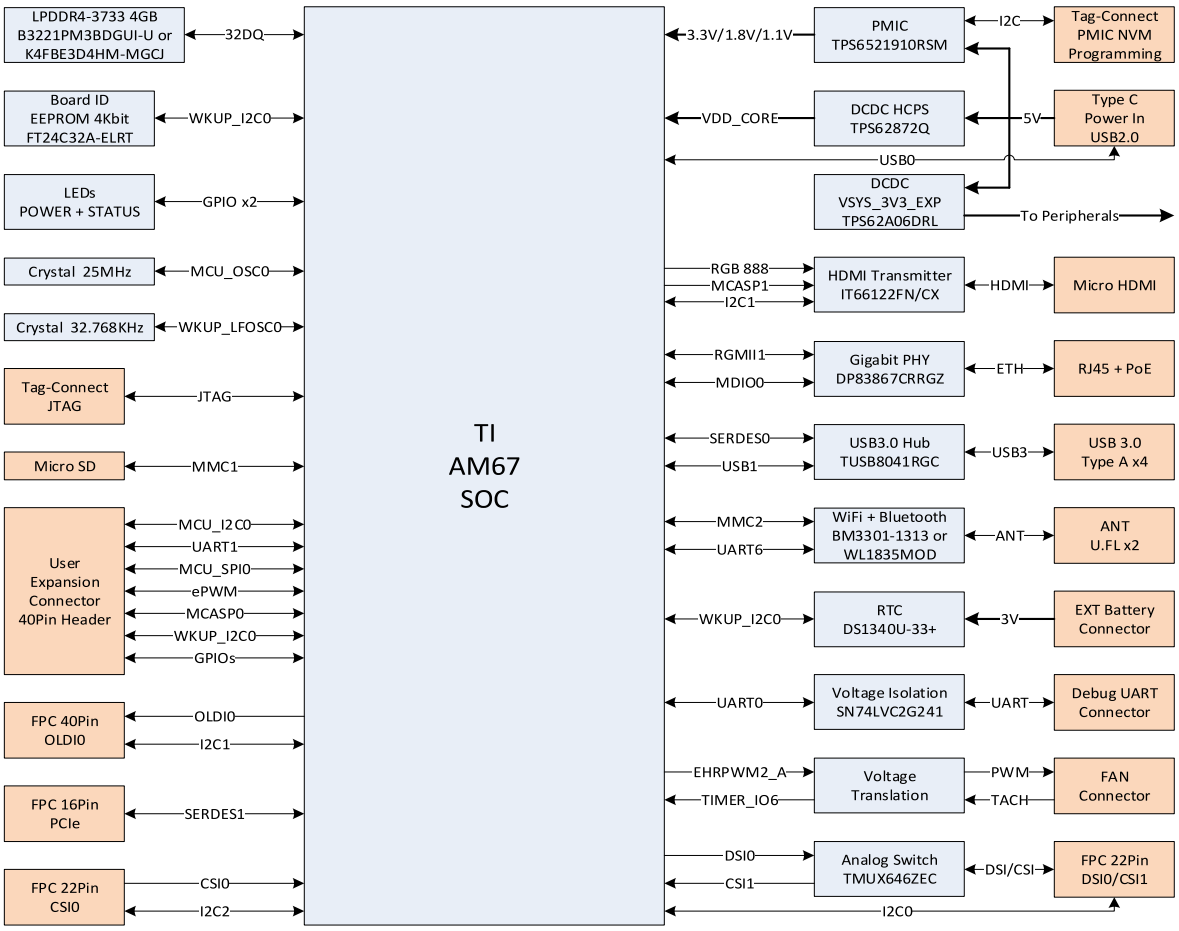


Fig. 3.1: BeagleY-AI block diagram

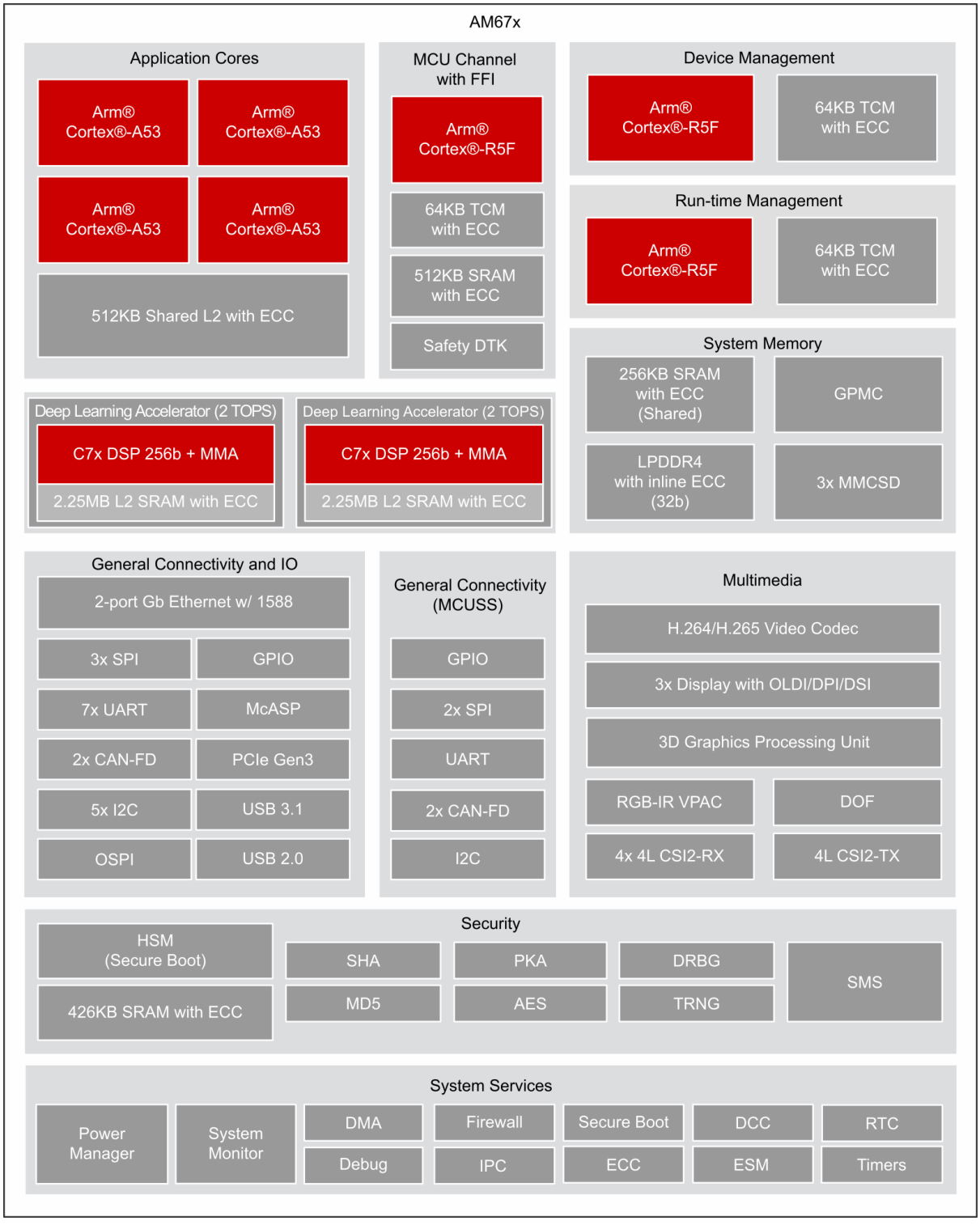


Fig. 3.2: AM67A block diagram

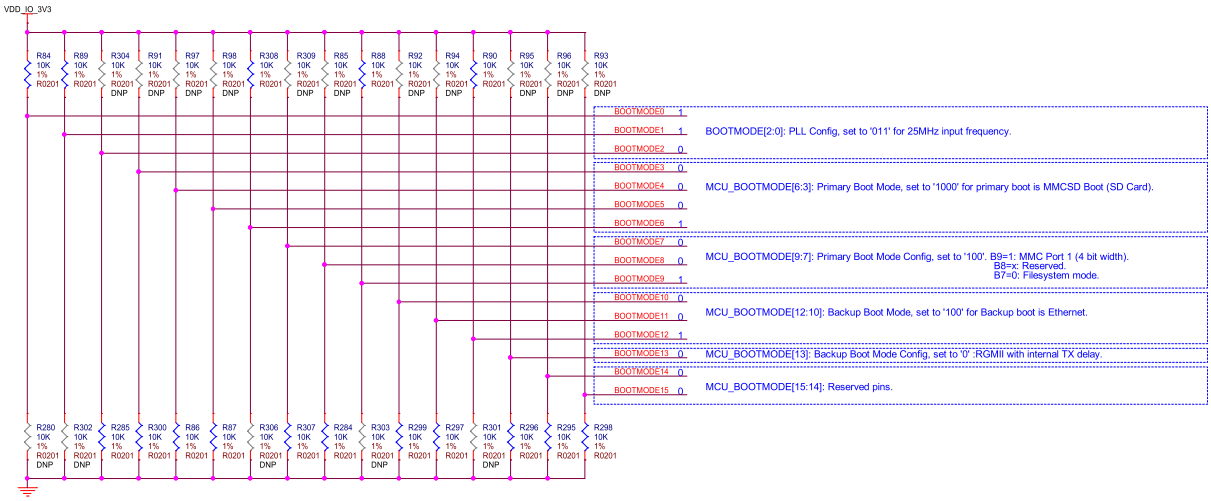


Fig. 3.3: BeagleY-AI boot modes

It is also possible to load U-Boot from the SD card and then load your main file system from another source, such as *Booting from NVMe Drives*.

3.4 Power

BeagleY-AI's power architecture is split between the TPS65219 PMIC which handles the main logic rails and a dedicated TPS62872 high current buck regulator for the SoC core rail which defaults to 0.85V on boot.

Both PMIC and VDD_CORE regulators are highly configurable but will boot the board to "sane" defaults out of box. For advanced users, it is possible to adjust both the VDD_CORE rail as well as IO rails (voltages, timings, behavior, etc.) for applications such as low power modes where you may want to trade clock speeds for power efficiency by running the SoC Core at 0.75V for example. Be careful, as changes here could result in unexpected behavior, the board not booting or even hardware damage, so tread carefully.

Note: At the time of writing, dynamic voltage switching is not supported by the AM67A SoC.

3.5 Clocks and Resets

BeagleY's main clock source is a 25Mhz Crystal Oscillator connected to MCU_OSC0 pins.

A 32.768Khz "Slow Clock" Crystal is used on the WKUP_LFOSC0 domain.

3.5.1 USB-C Power/Data Port

The board is primarily intended to be powered via USB-C. PD Power negotiation is not done dynamically but rather by tying the CC lines to GND via 5.1KΩ resistors to indicate to the PD Source that the device requires 5V 3A. Using USB-PD power supplies rated for higher wattages is safe as they will always negotiate to the 5V 3A requested by the board.

The USB-C port is configured by default to also show up as a USB2.0 Device which exposes a serial console, ethernet gadget (for connection sharing) as well as MTP (Flash Drive) so that only one cable is required to use the board. A Type-C to Type-C cable and avoiding un-powered USB hubs is recommended due to the board's power consumption requirements. Inadequate behavior may result in brownouts/resets or other unexpected behavior.

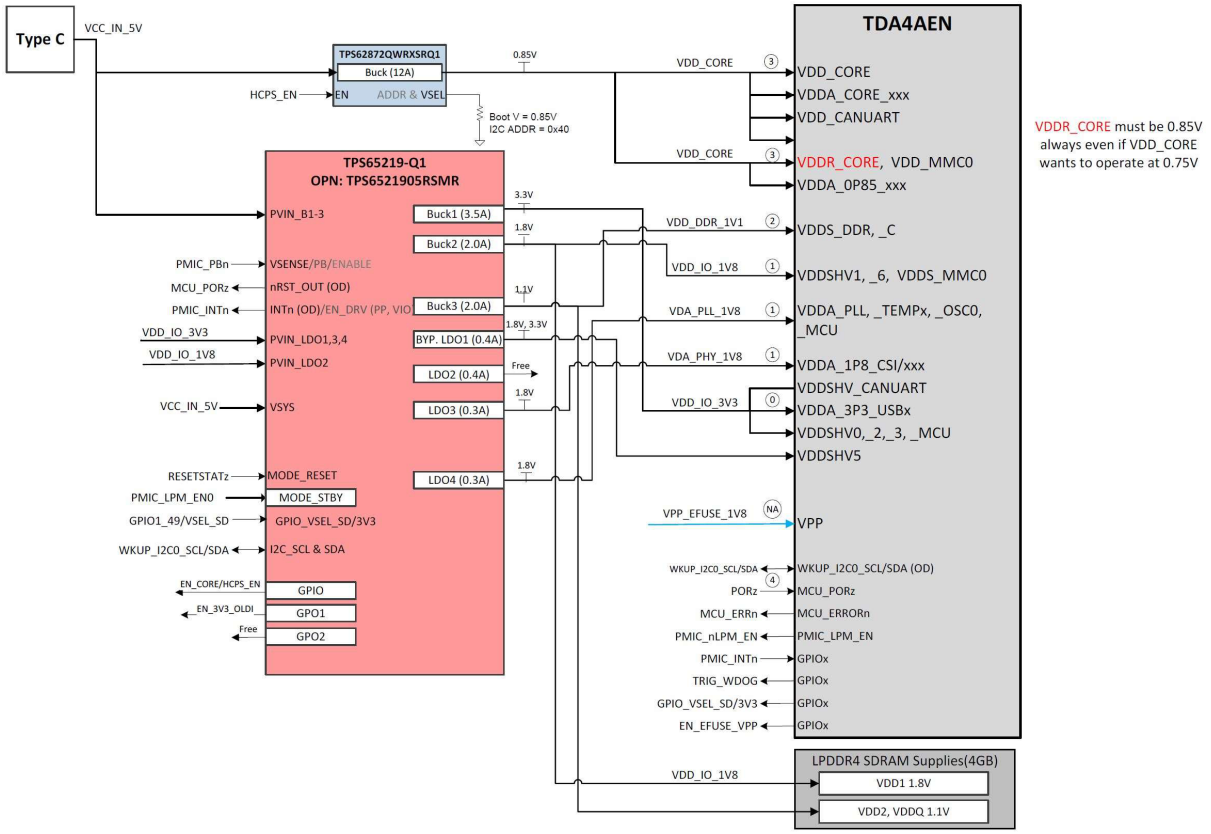


Fig. 3.4: BeagleY-AI power distribution network

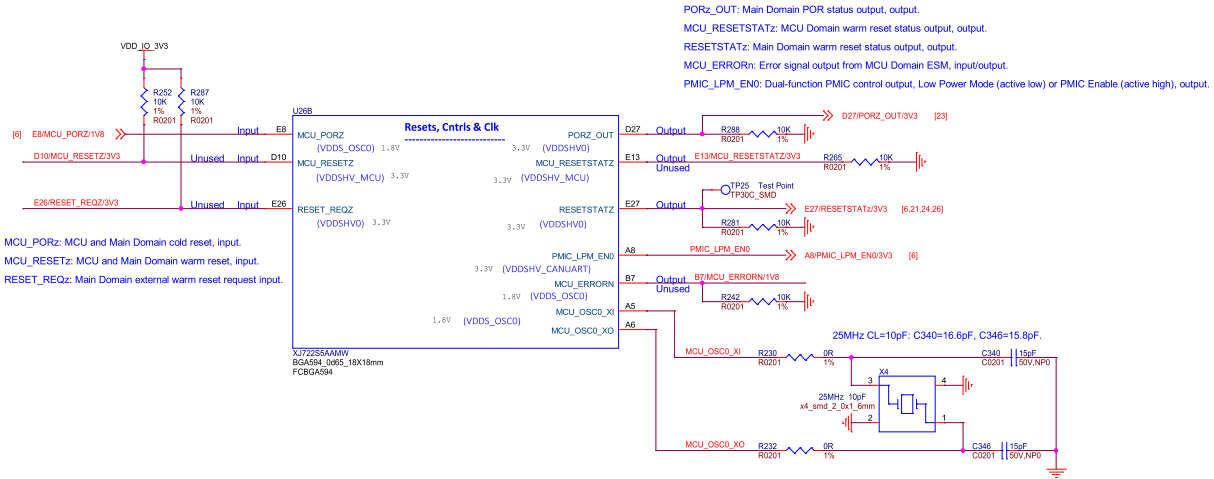


Fig. 3.5: BeagleY-AI SoC Reset, Cntrls, and Clk

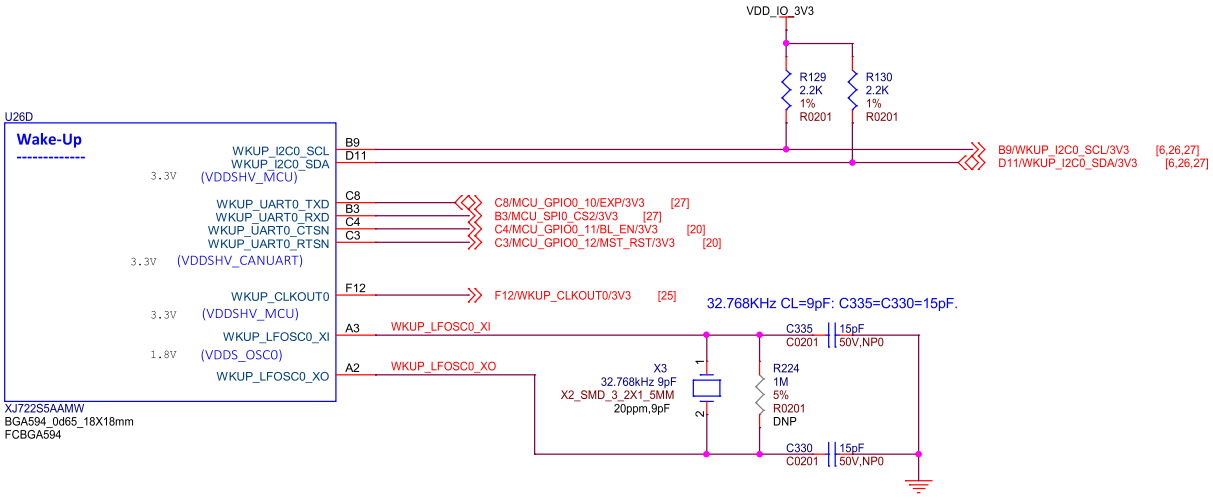


Fig. 3.6: BeagleY-AI wkup reset cntrls osc

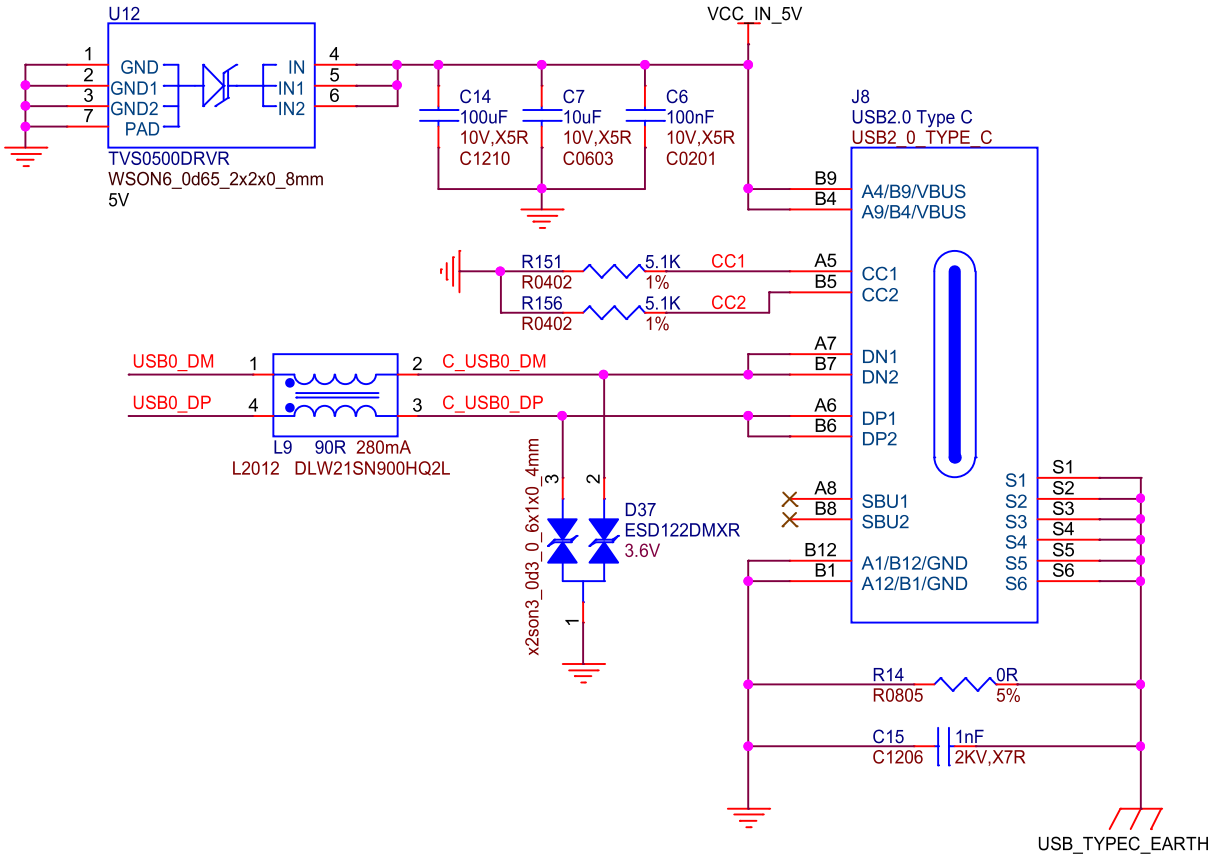


Fig. 3.7: BeagleY-AI USB-C

3.5.2 PMIC

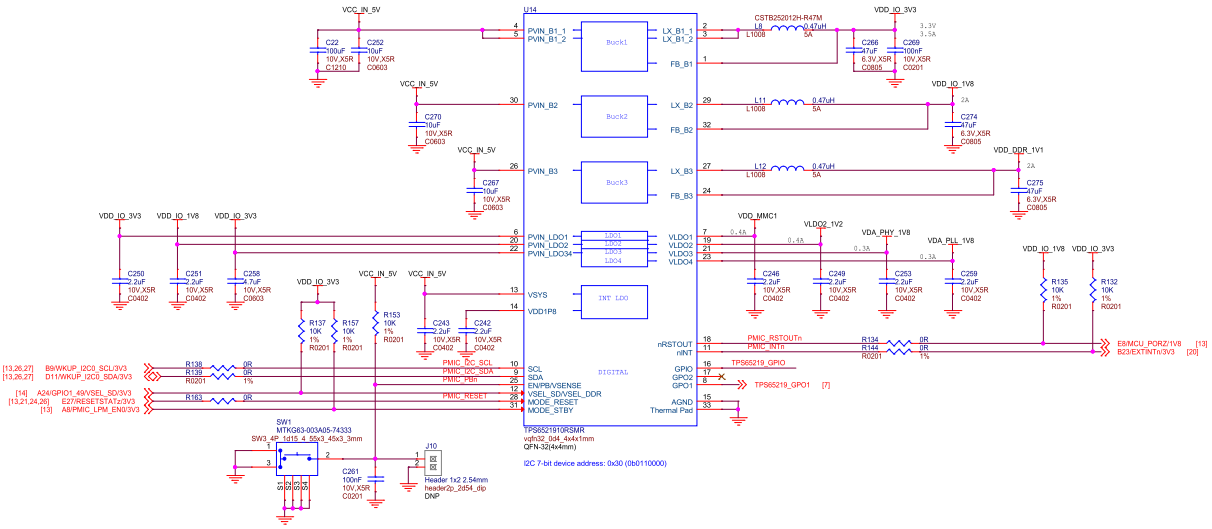


Fig. 3.8: BeagleY-AI PMIC

3.5.3 HCPS (High Current Power Stage)

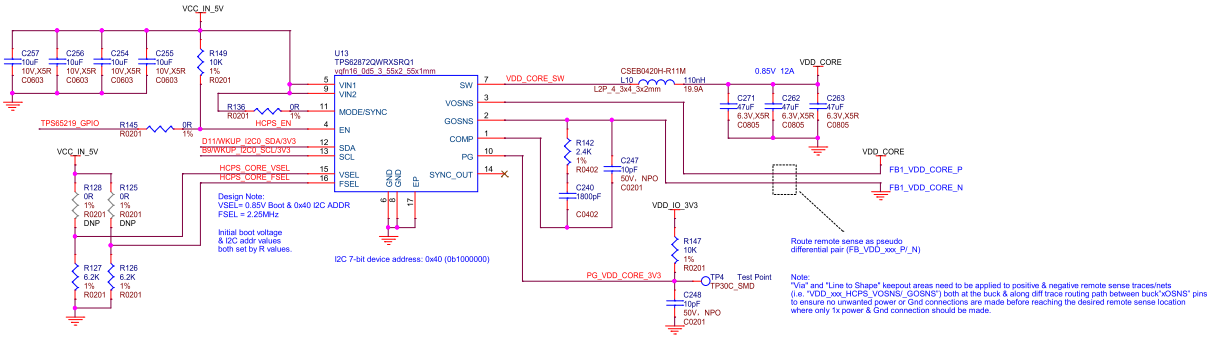


Fig. 3.9: BeagleY-AI VDD core High Current Power Stage (HCPS)

3.5.4 Analog Rail Decoupling

3.5.5 Digital Rail Decoupling

Note: Other power sections are nested within their specific interface section.

3.5.6 LDOs

While the 3.3V VDD_IO rail is provided by the PMIC, the actual “high current” VSYS 3.3V rail used on the expansion header and elsewhere in the system is provided by a discrete TPS62A06DRLR regulator.

The 2V5 Rail used by the Ethernet PHY is generated a discrete TPS74801 regulator. This regulator is fed by the 3V3 VSYS regulator.

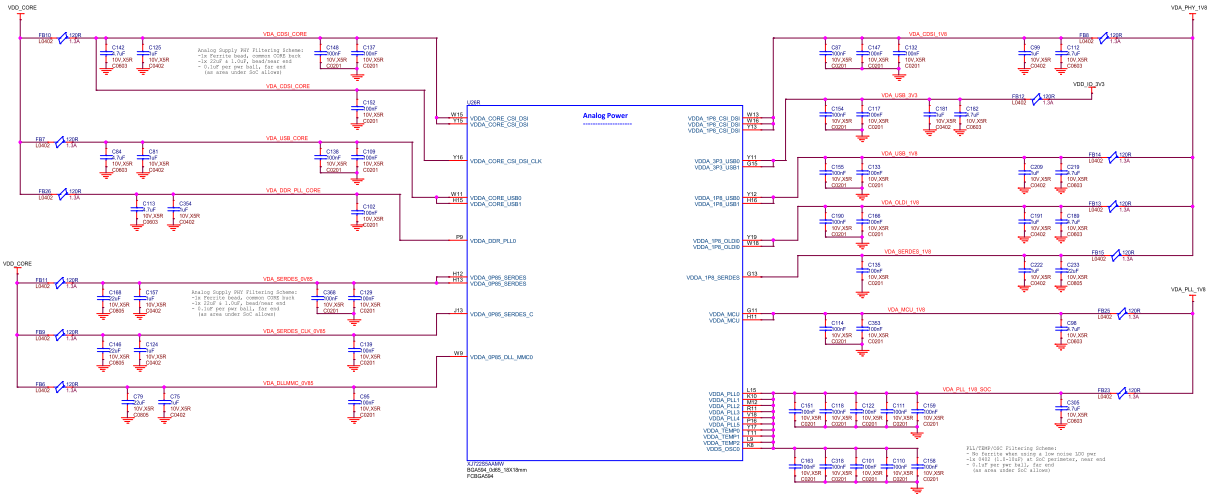


Fig. 3.10: BeagleY-AI SoC analog power rail decoupling capacitors

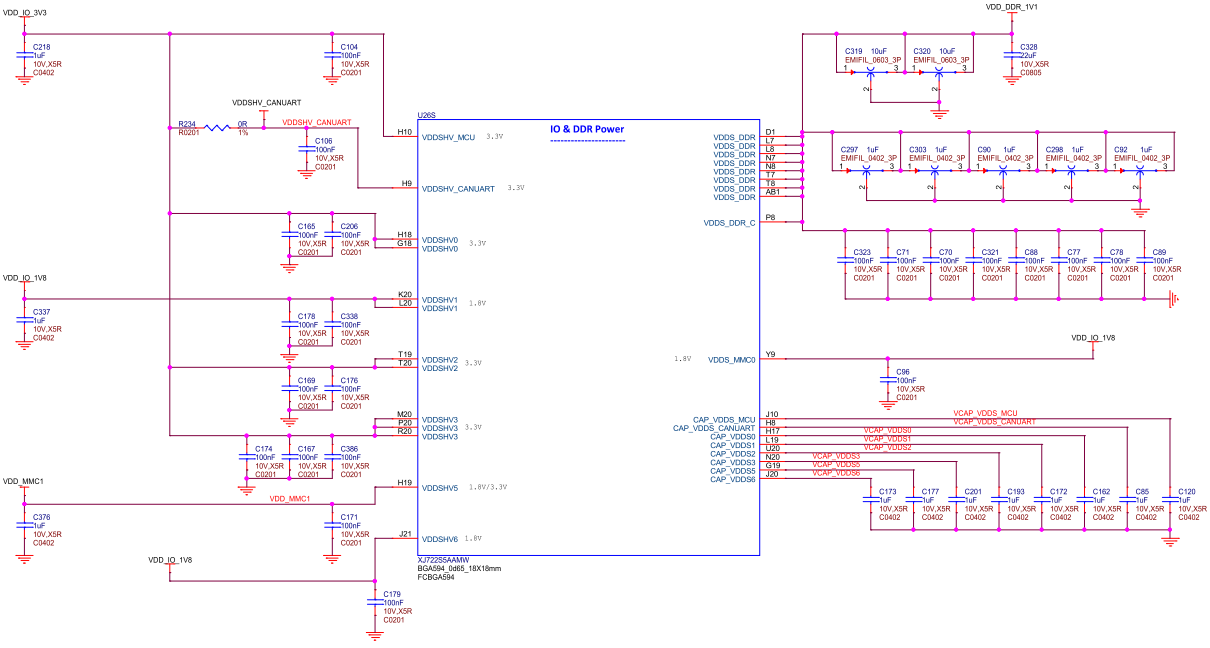


Fig. 3.11: BeagleY-AI AI SoC IO and DDR decoupling capacitors

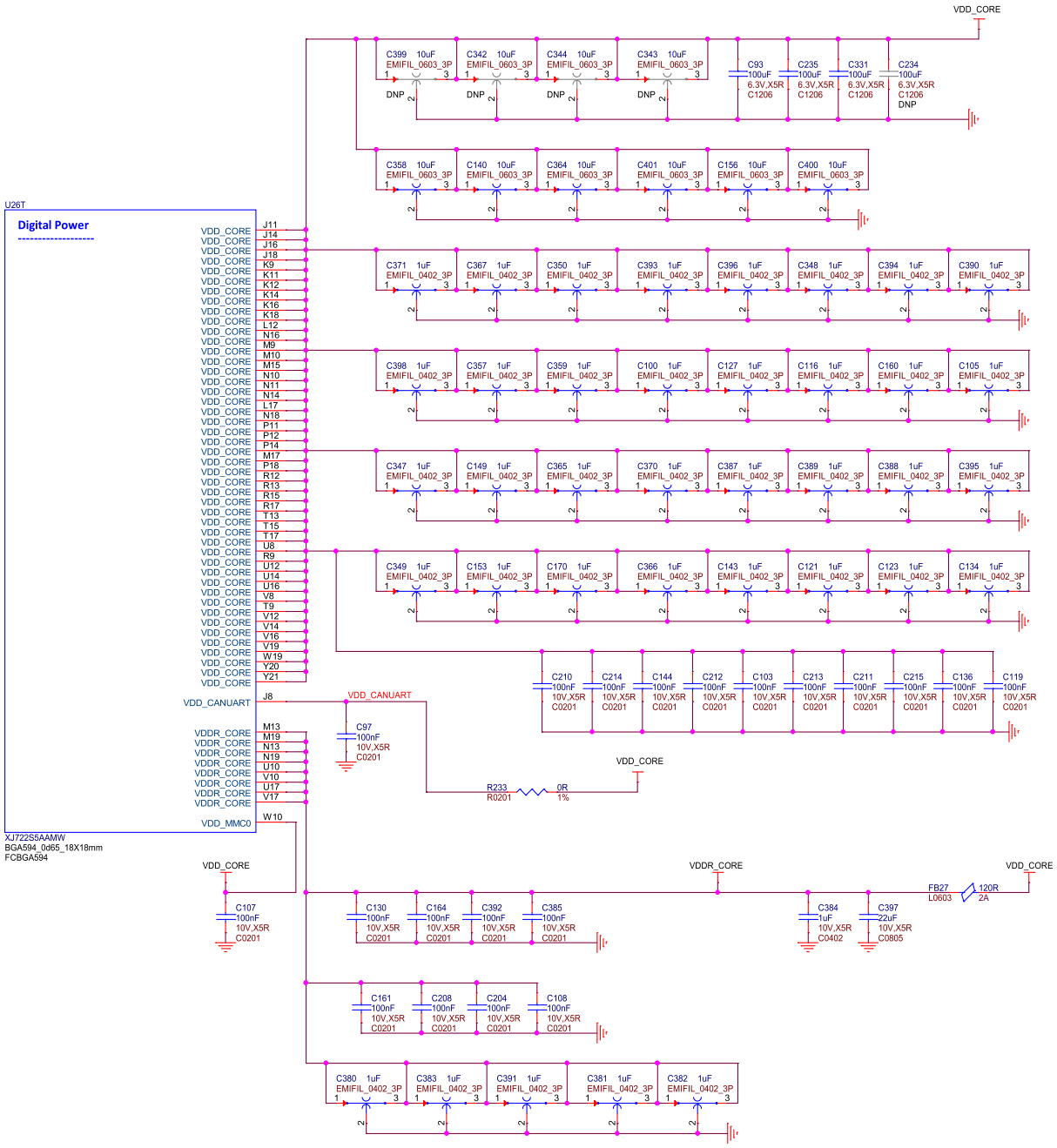


Fig. 3.12: BeagleY-AI SoC VDD & VDDR_CORE decoupling capacitors

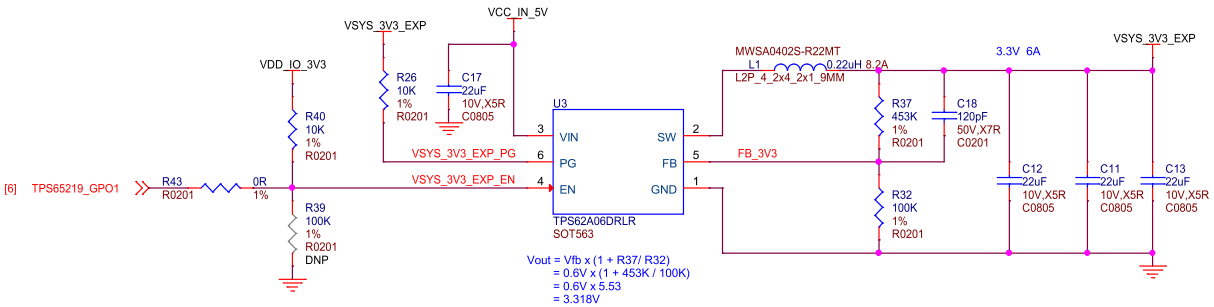


Fig. 3.13: BeagleY-AI VSYS 3V3

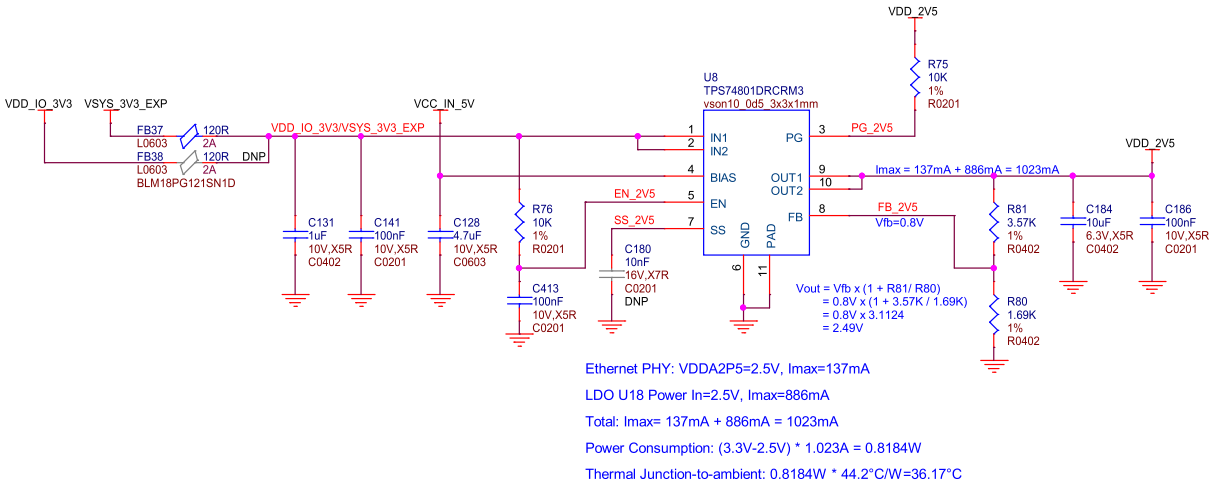


Fig. 3.14: BeagleY-AI ethernet power 3V3 to 2V5

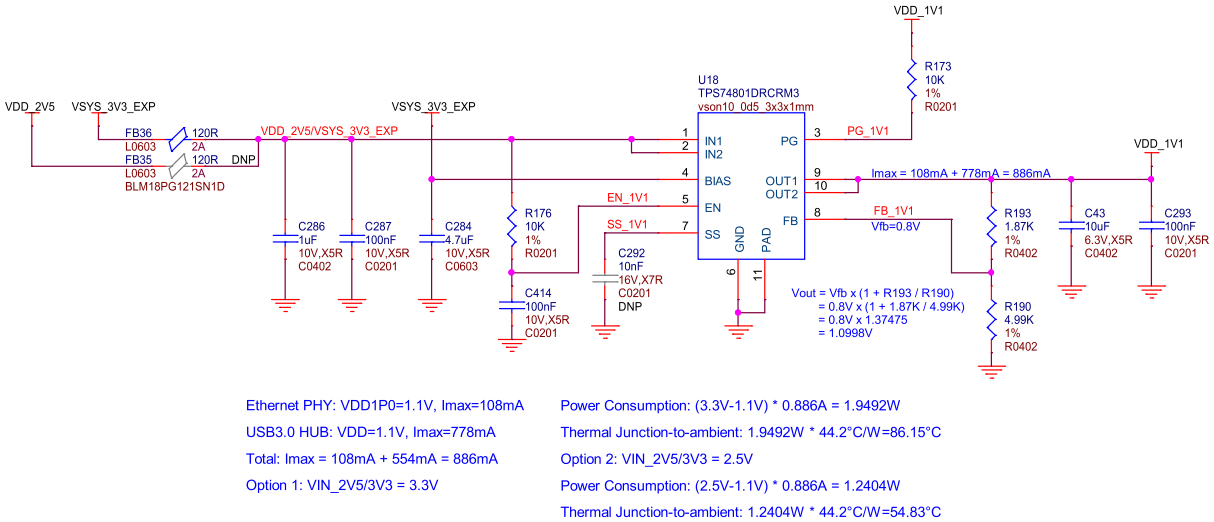


Fig. 3.15: BeagleY-AI 3V3/V5 to 1V1 LDO

The 1V1 Rail used by the PHY and USB 3.1 Hub is generated a discrete TPS74801 regulator. By default, this regulator is fed by the 3V3 VSYS regulator previously discussed.

3.6 Memory

3.6.1 RAM (LPDDR4)

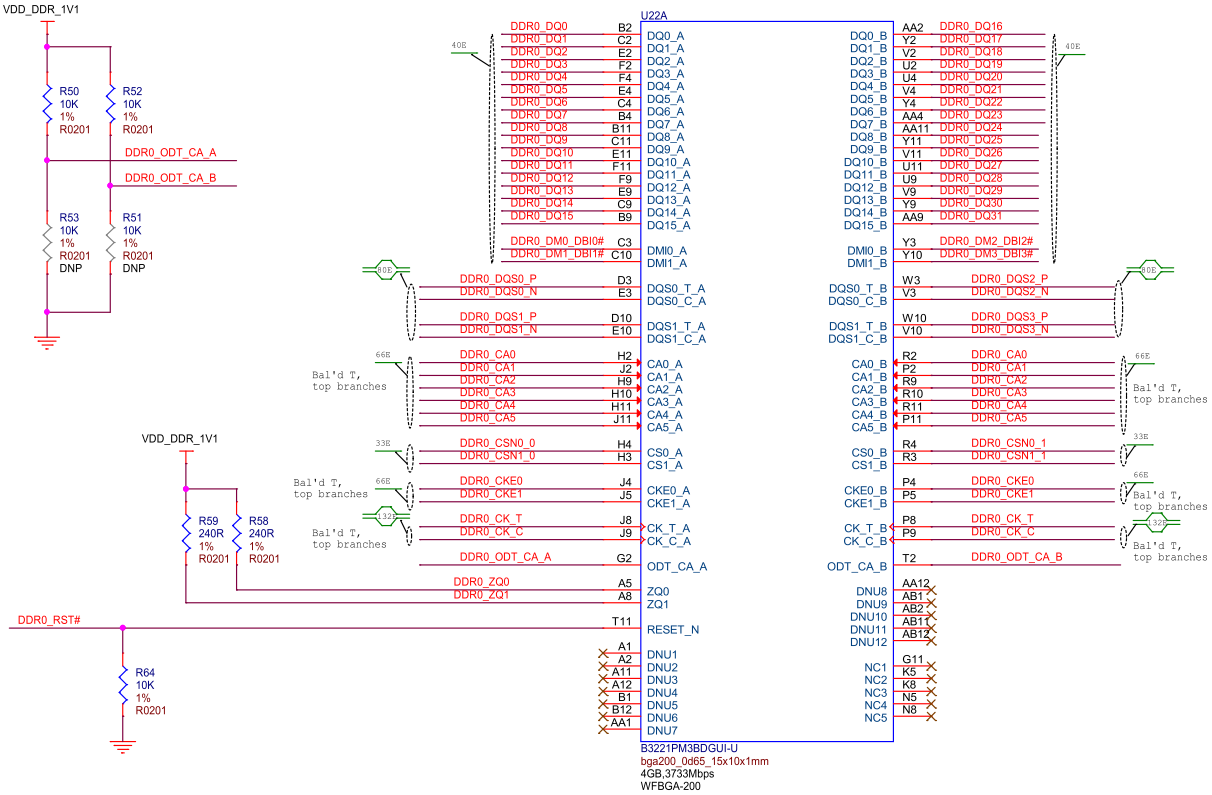


Fig. 3.16: BeagleY-AI DDR

BeagleY-AI has 4GB of Kingston x32 LPDDR4 Memory.

Todo: Add Final DDR Part Number

3.6.2 EEPROM

BeagleY-AI features an on-board FT24C32A 32Kbit I2C EEPROM for storing things like board information, manufacture date, etc.

Todo: Add details about specific EEPROM contents and formatting.

3.6.3 microSD Card

The microSD card is the primary boot interface for BeagleY-AI, it corresponds to the MMC1 interface on the AM67A SoC.

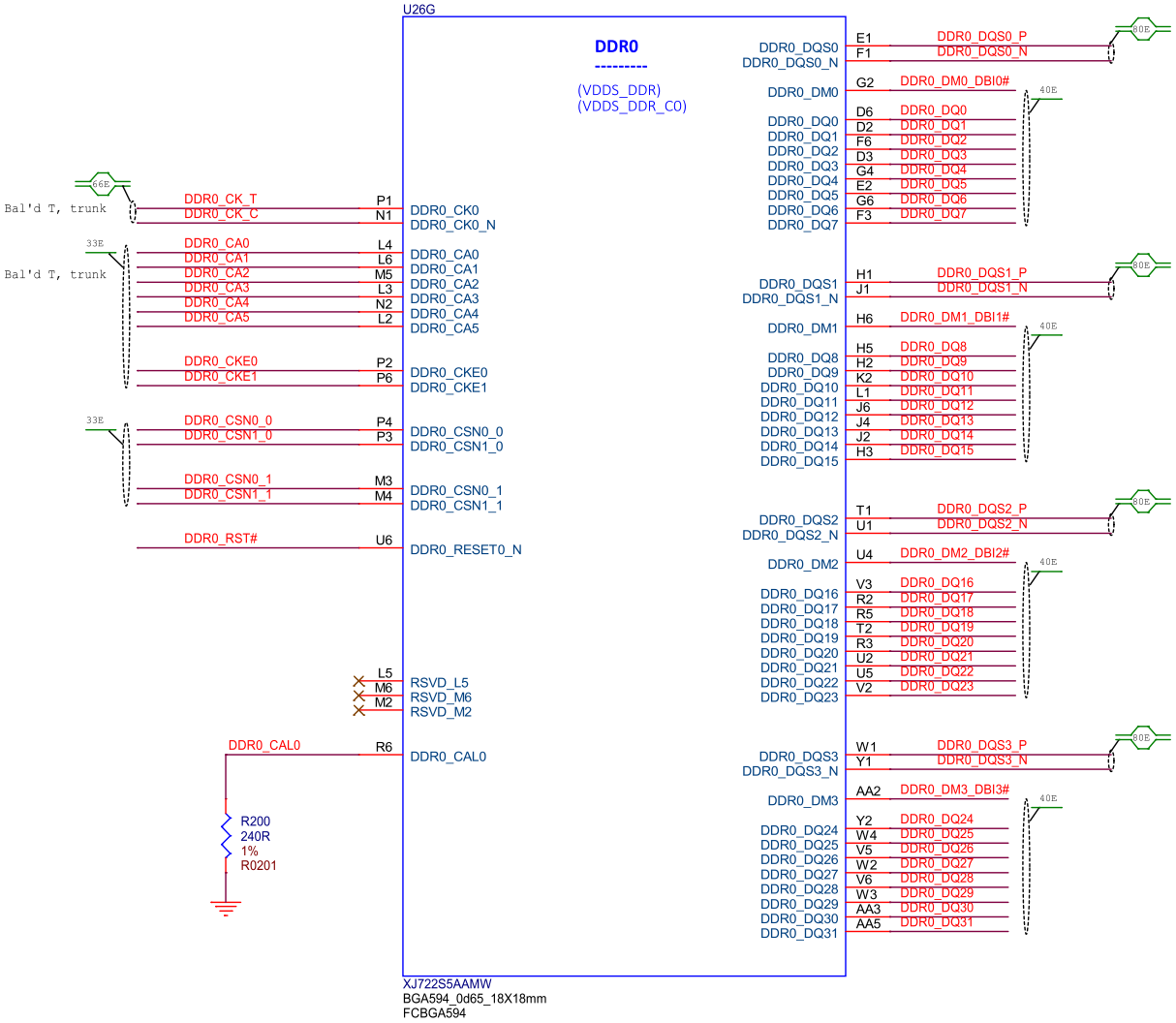


Fig. 3.17: BeagleY-AI SoC DDR0 connections

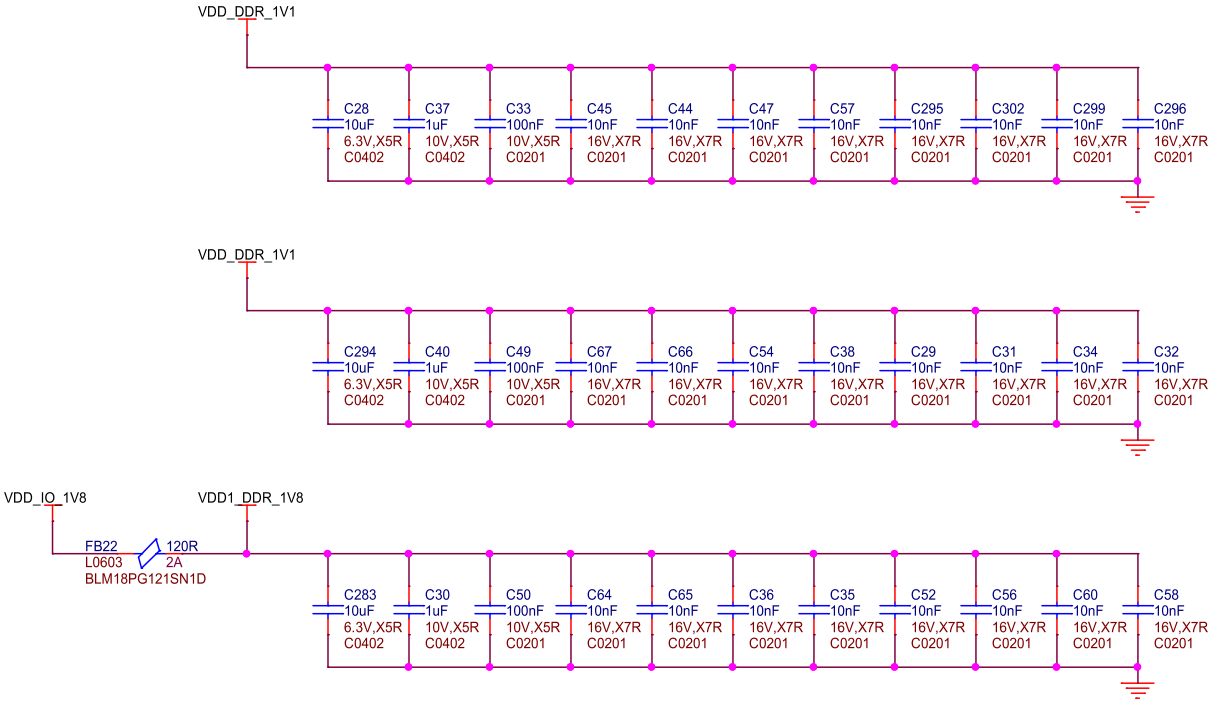


Fig. 3.18: BeagleY-AI DDR caps

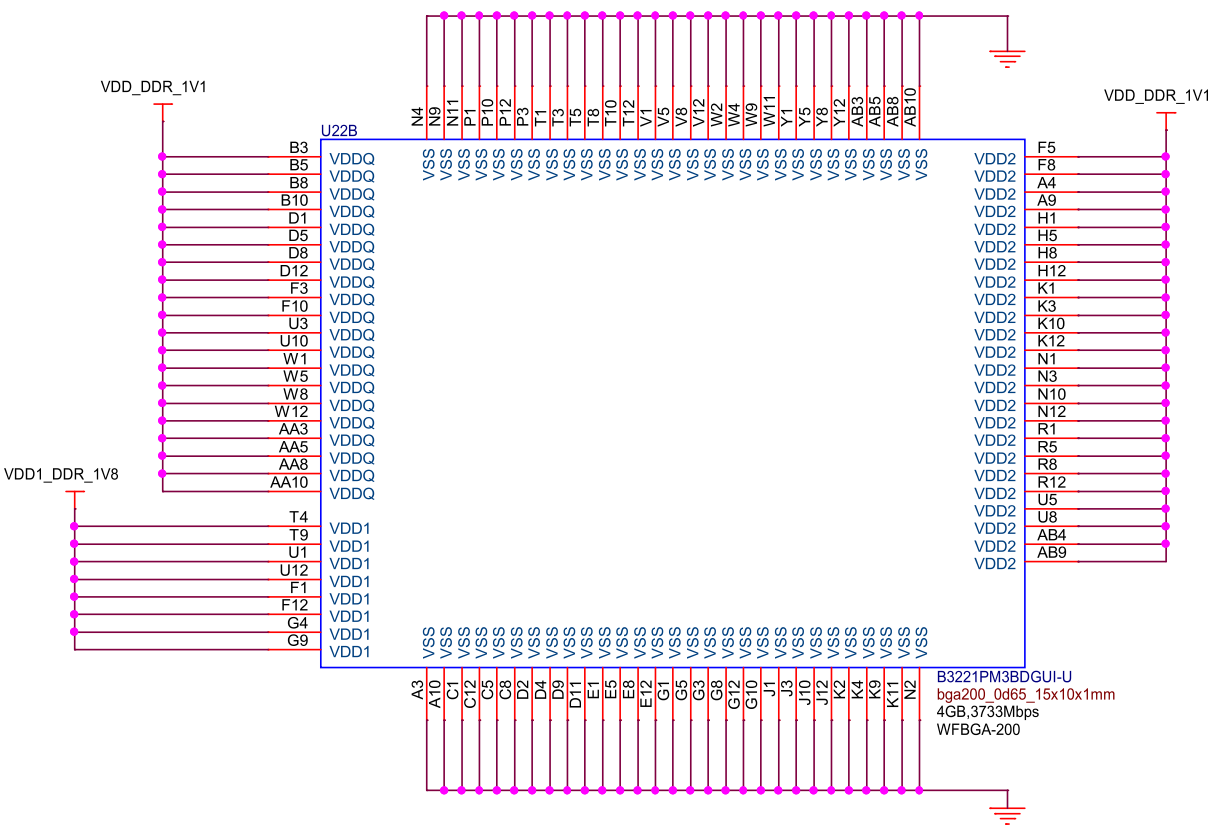


Fig. 3.19: BeagleY-AI DDR power

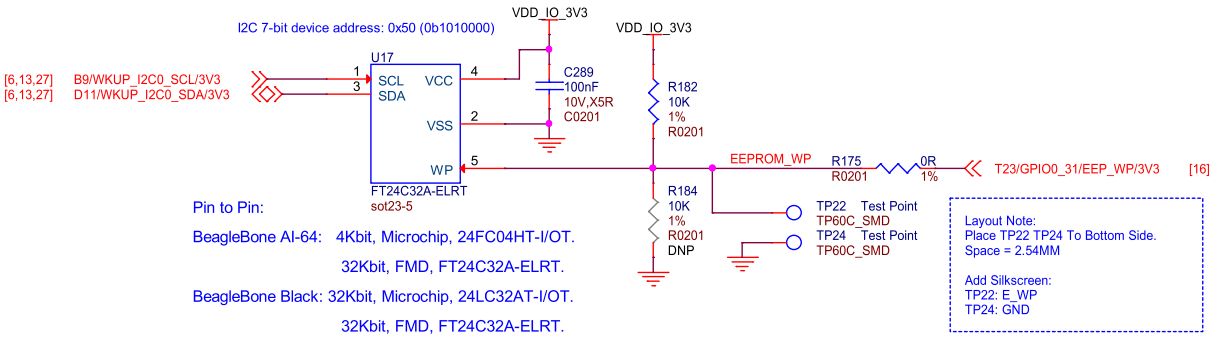


Fig. 3.20: BeagleY-AI board id eeprom

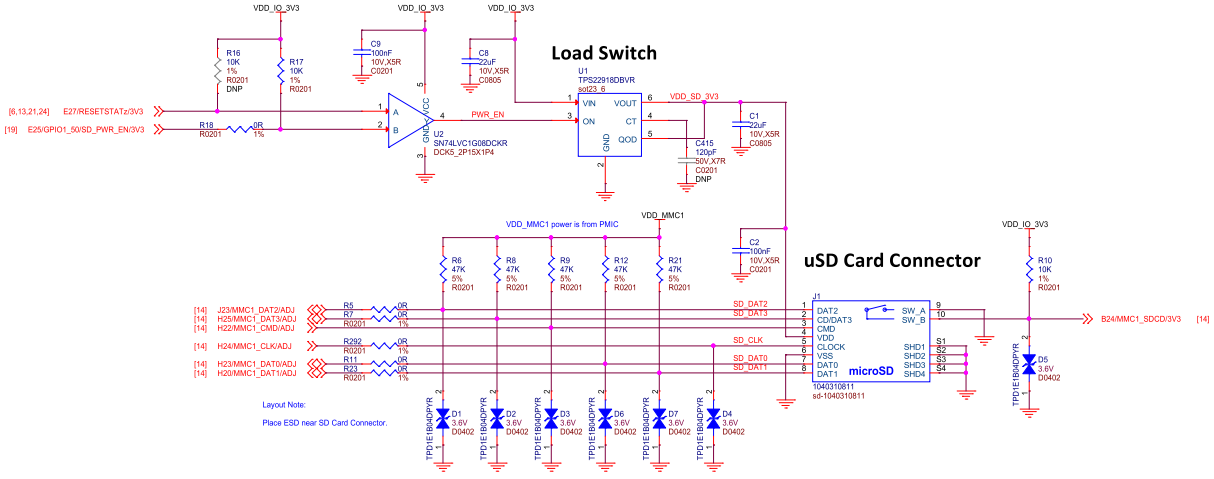


Fig. 3.21: BeagleY-AI microSD card interface

To enable UHS-1 SD card functionality (and speeds!), a load switch is provided which allows the SoC MMC1 PHY to switch the SD Card IO voltage to 1.8V.

Todo: Explain UHS-1 in more detail and add link to TRM for boot modes and resistor swap options for advanced users.

3.7 General Expansion

3.7.1 40pin Header

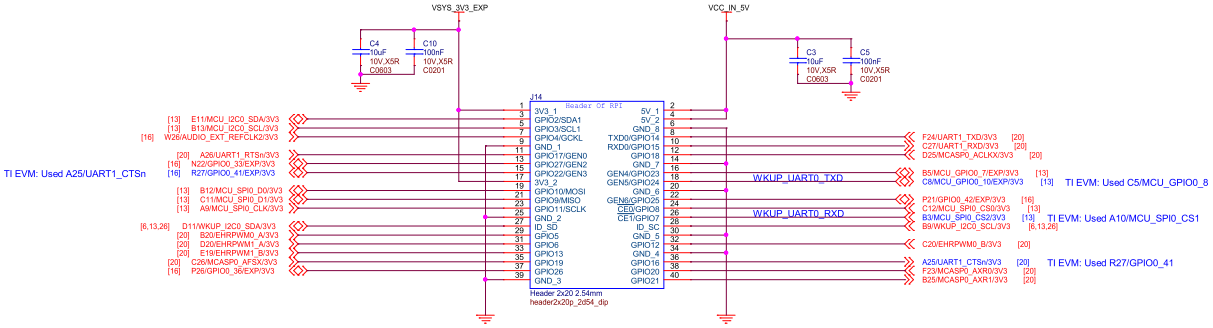


Fig. 3.22: BeagleY-AI user expansion connector

BeagleY-AI features a 40-pin GPIO Header which aims to enable compatibility with a lot of existing Raspberry Pi HAT add-on boards. See pinout.beagleboard.io for a more comprehensive view of the 40 pin GPIO header, available pin functions and tested accessories!

Todo: Add link to docs on building expansion accessories.

3.7.2 I2C

By default, 5 different I2C interfaces are exposed, all of which feature external 2.2KΩ pull-up resistors. 3 of the interfaces are used by the CSI, DSI and OLDI ports for Cameras & Displays. The remaining 2 ports are exposed on the 40pin GPIO expansion connector.

The MCU_I2C0 interface is intended as the primary external I2C interface for BeagleY-AI and matches physical pins 3 and 5 of the header. Most HATs will use these pins.

While WKUP_I2C0 is also exposed on the 40pin Header (physical pins 27 & 28), that bus is shared with several on-board devices, namely the PMIC, VDD_CORE regulator, Board ID EEPROM and RTC. As such, it is highly advisable to leave these pins unused unless you are sure you know what you are doing. These pins are normally only pinned out as a “HAT EEPROM detect” for RPi HATs that provide such functionality (of which there are very few)

See pinout.beagleboard.io/pinout/i2c for a more visual explanation.

3.7.3 USB

BeagleY-AI features a USB3.1 HUB that provides 4 total USB3.1 Ports from a single USB3.1 Gen-1 (5 Gbps) SERDES0 lane.

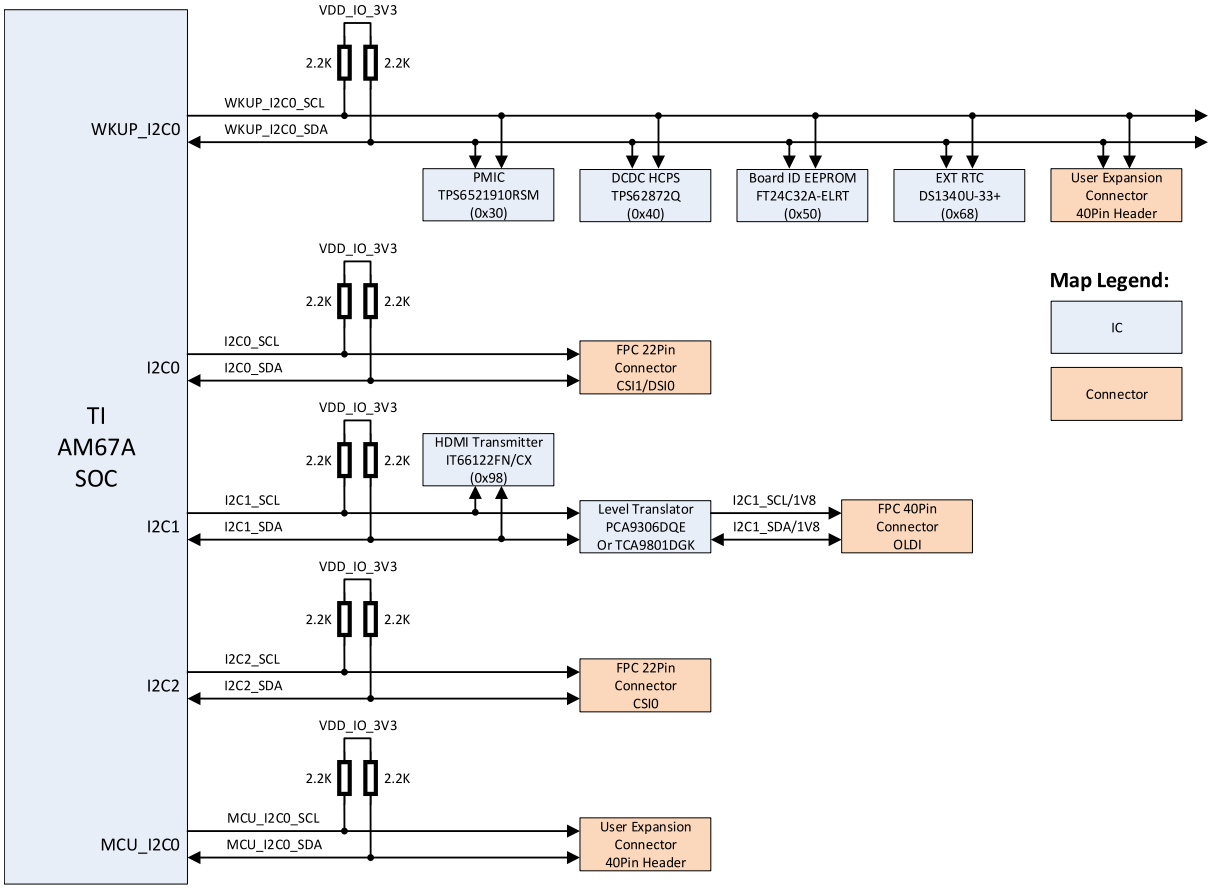


Fig. 3.23: BeagleY-AI I2C tree

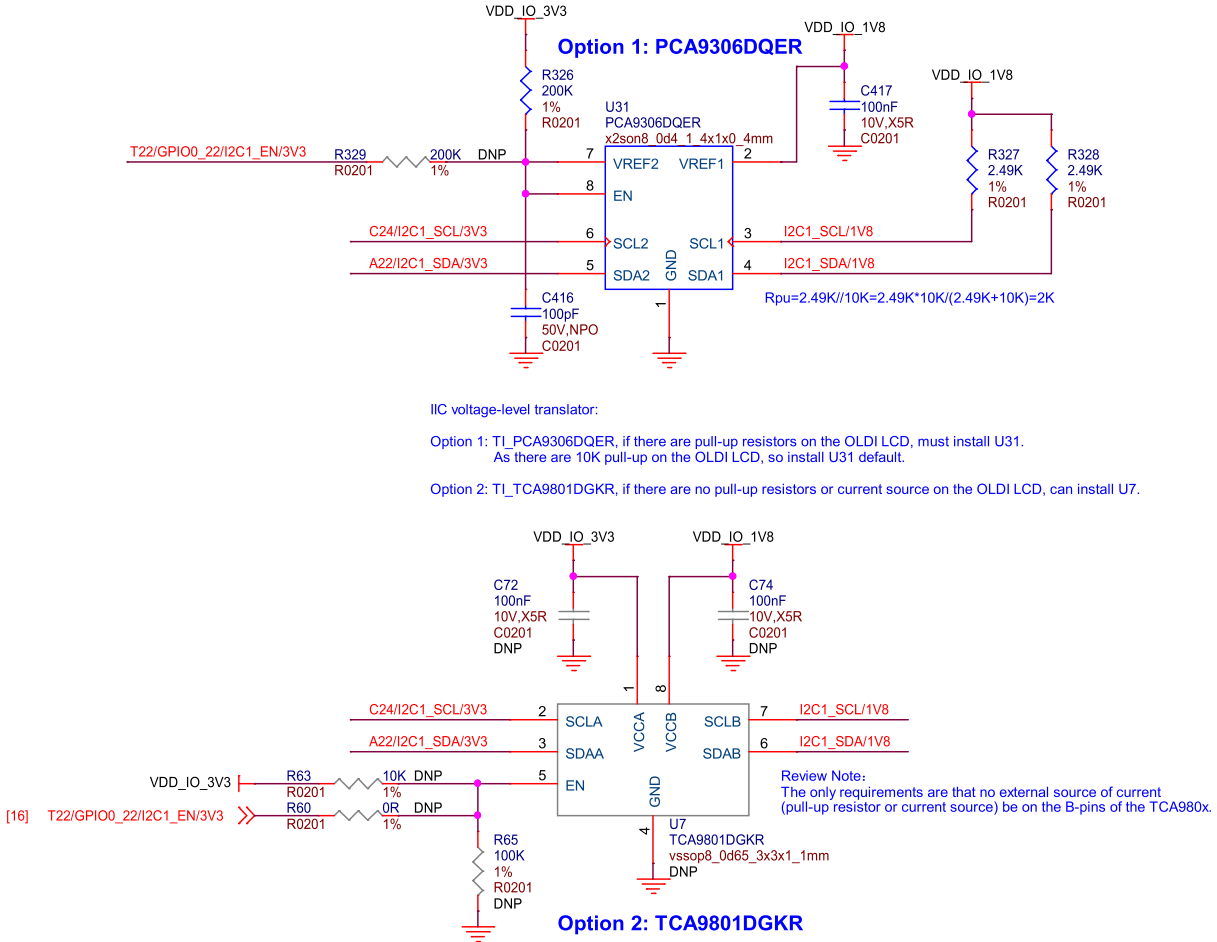


Fig. 3.24: BeagleY-AI voltage level translator

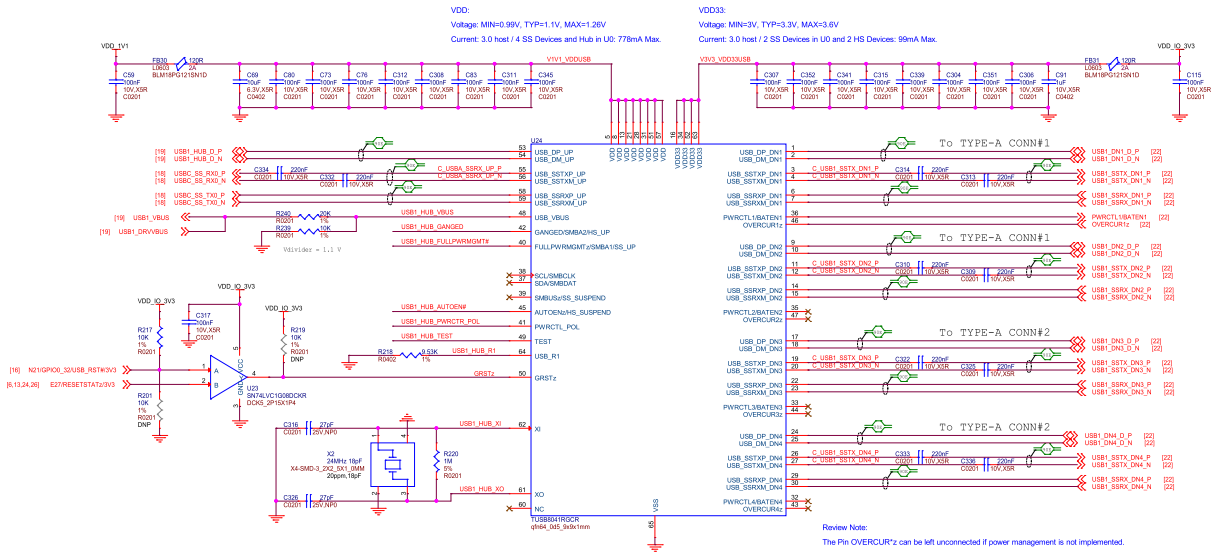


Fig. 3.25: BeagleY-AI USB3 hub

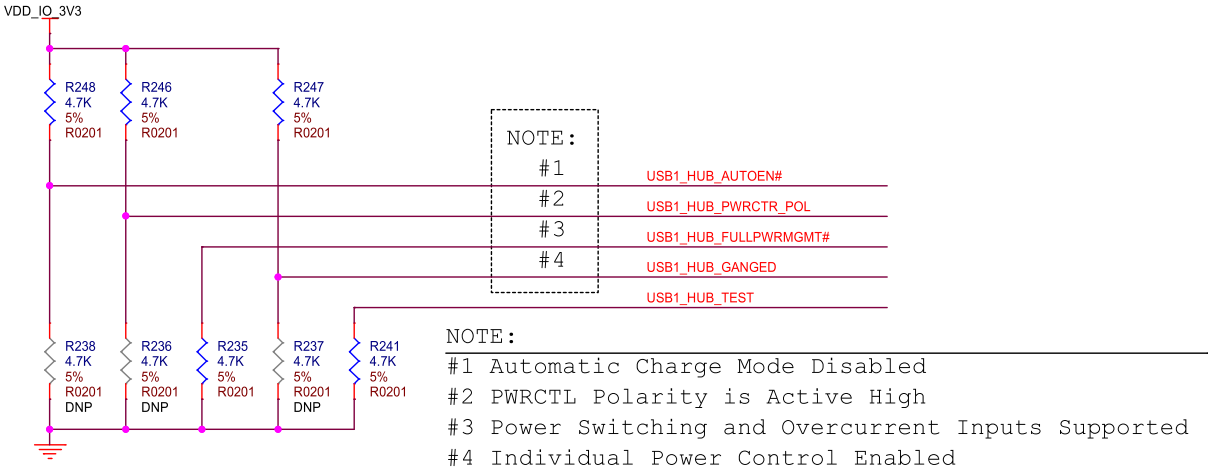


Fig. 3.26: BeagleY-AI USB hub config

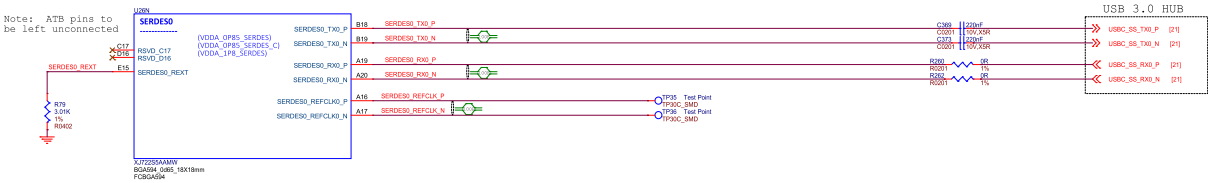


Fig. 3.27: BeagleY-AI SoC SERDES0

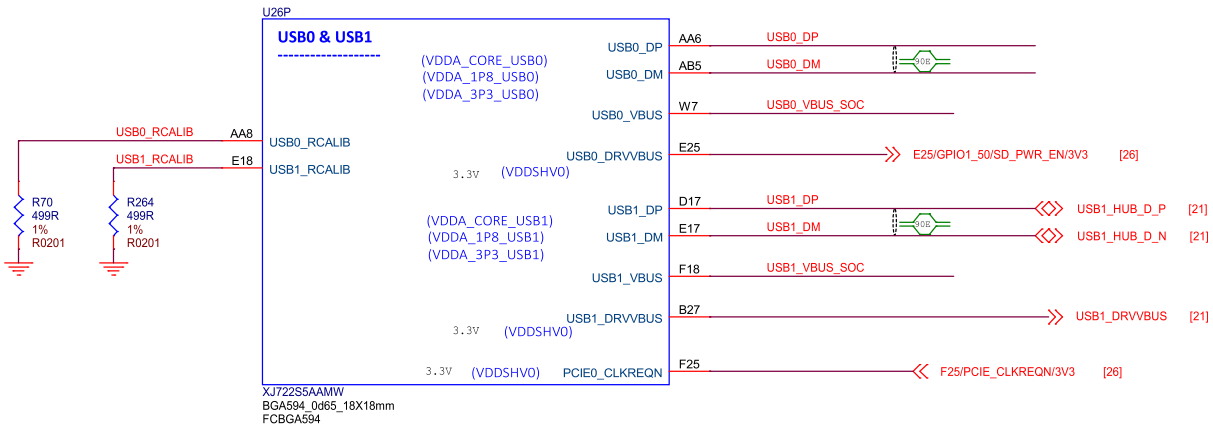


Fig. 3.28: BeagleY-AI SoC USB0 and USB1

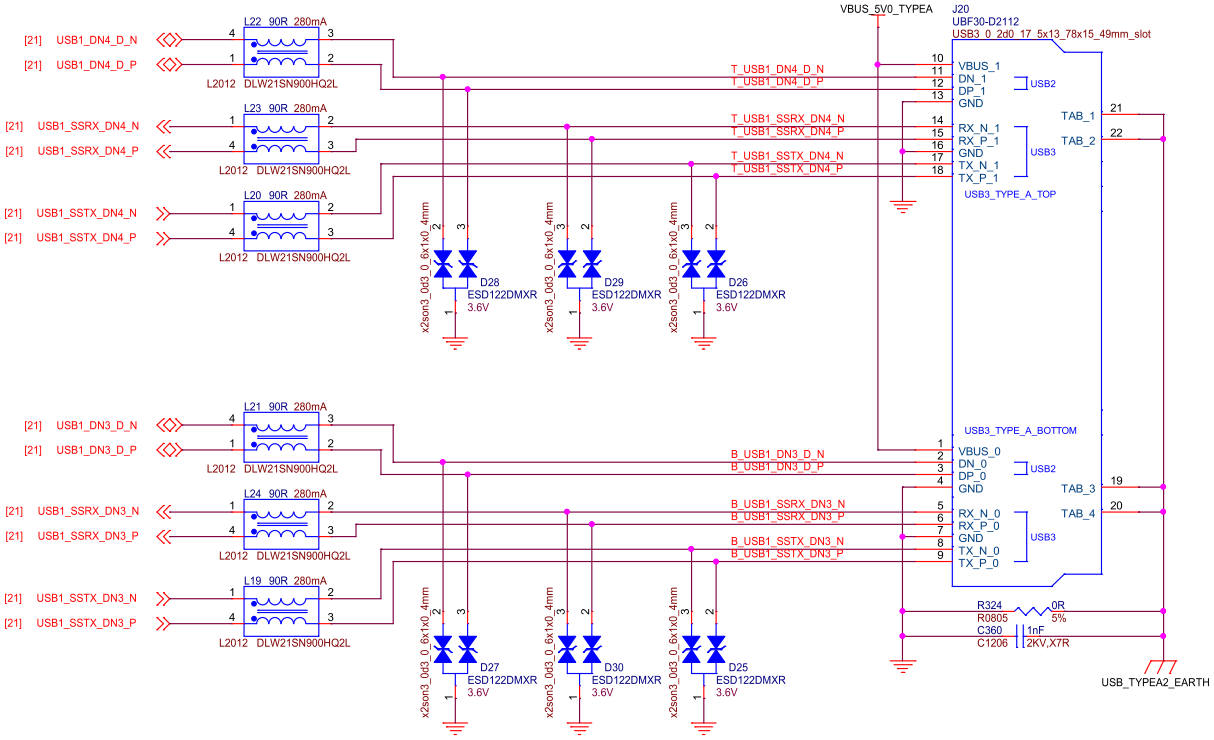


Fig. 3.29: BeagleY-AI USB-A Connector 1

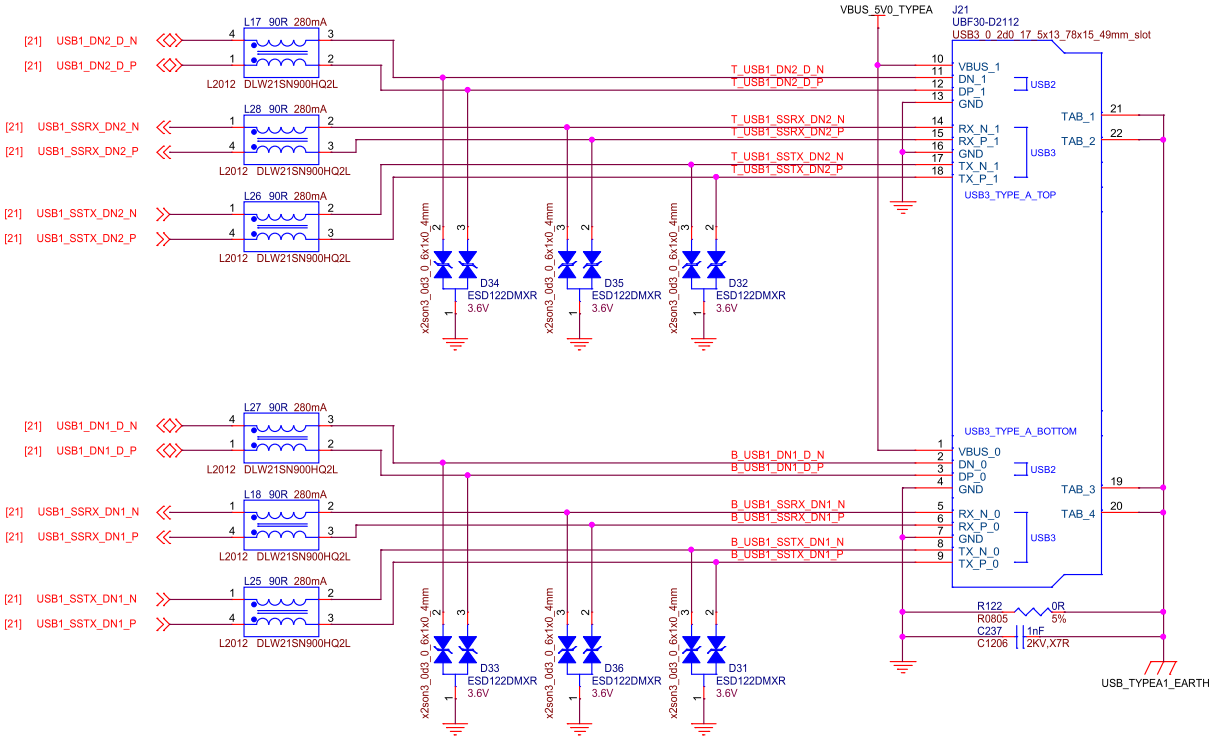


Fig. 3.30: BeagleY-AI USB-A Connector 2

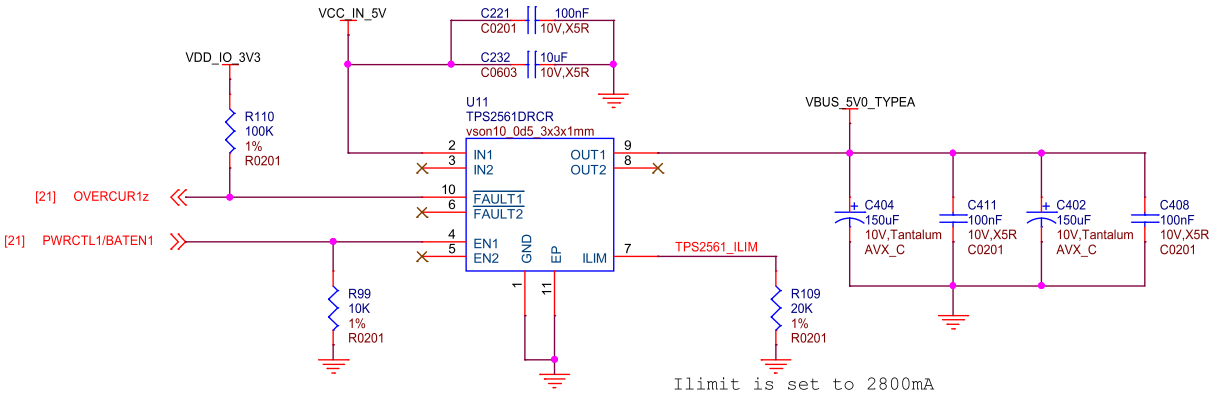


Fig. 3.31: BeagleY-AI dual USB current limiter

BeagleY-AI features a dedicated USB current limiter that will prevent the Type-A ports from drawing power in excess of 2.8A.

3.7.4 PCI Express

BeagleY-AI features an RPi 5 compatible PCIe connector rated for PCIe Gen2 x1 (5GT/s) connected to SERDES1 on AM67A.

Note: Just like the Raspberry Pi 5, while the AM67A SoC is capable of PCIe Gen3 (8GT/s), the choice of cable/connector means that some devices may not be able to run at full Gen 3 speeds and will need to be limited to Gen 2 for stable operation.

3.7.5 RTC (Real-time Clock)

BeagleY-AI has an on-board I2C RTC that can be powered by an external RTC for accurate time-keeping even when the board is powered off. For more information, see the corresponding docs page - *Using the on-board Real Time Clock (RTC)*

3.7.6 Fan Header

BeagleY-AI features a Raspberry Pi 5 compatible Fan connector. The fan is software PWM controller in Linux by default to maintain a balance between cooling and noise depending on SoC temperature.

3.8 Networking

3.8.1 WiFi / Bluetooth LE

BeagleY-AI features a Beagle BM3301 Wireless module based on the Texas Instruments CC3301 which features 2.4Ghz WiFi6 (802.11AX) and BLE 5.4

Note: 5Ghz WiFi Bands and Bluetooth Classic are not supported by the CC3301.

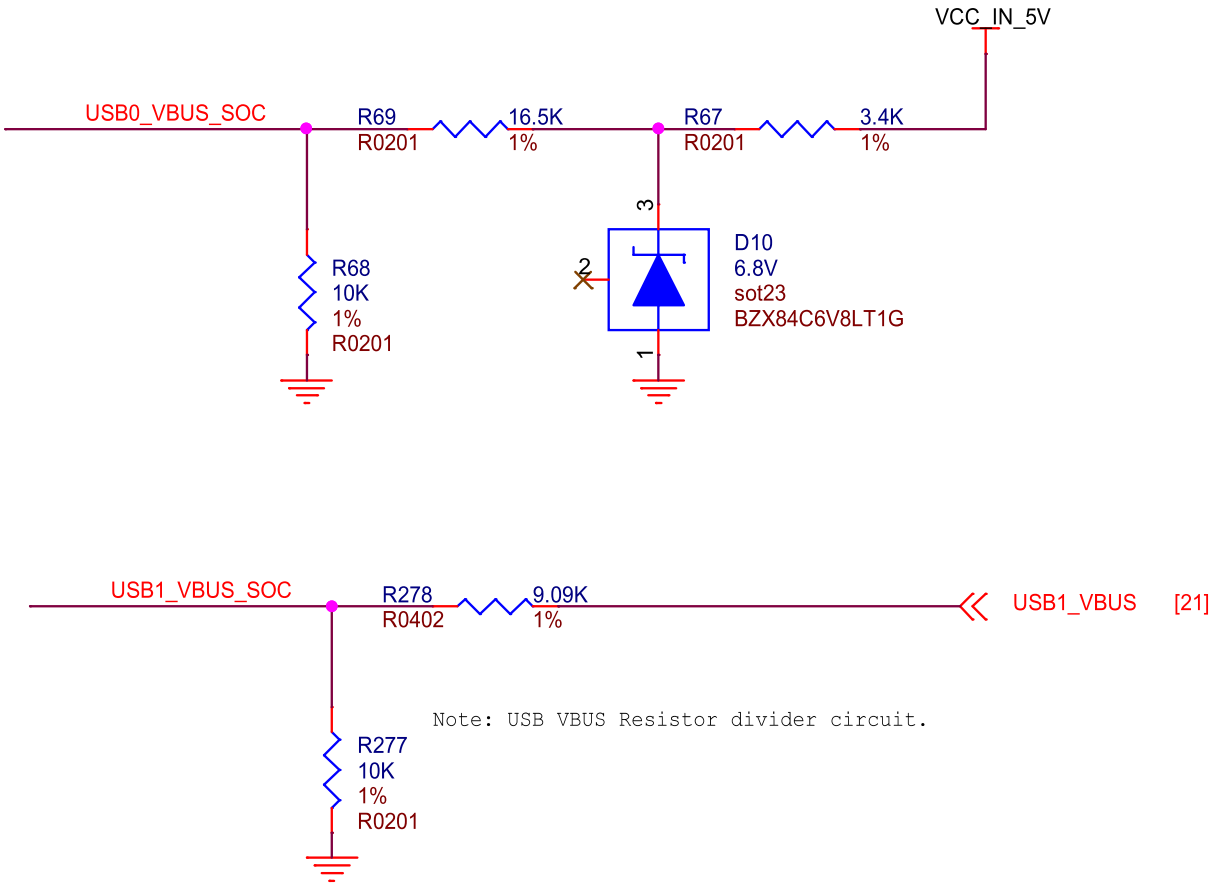


Fig. 3.32: BeagleY-AI USB VBUS resistor divider circuit

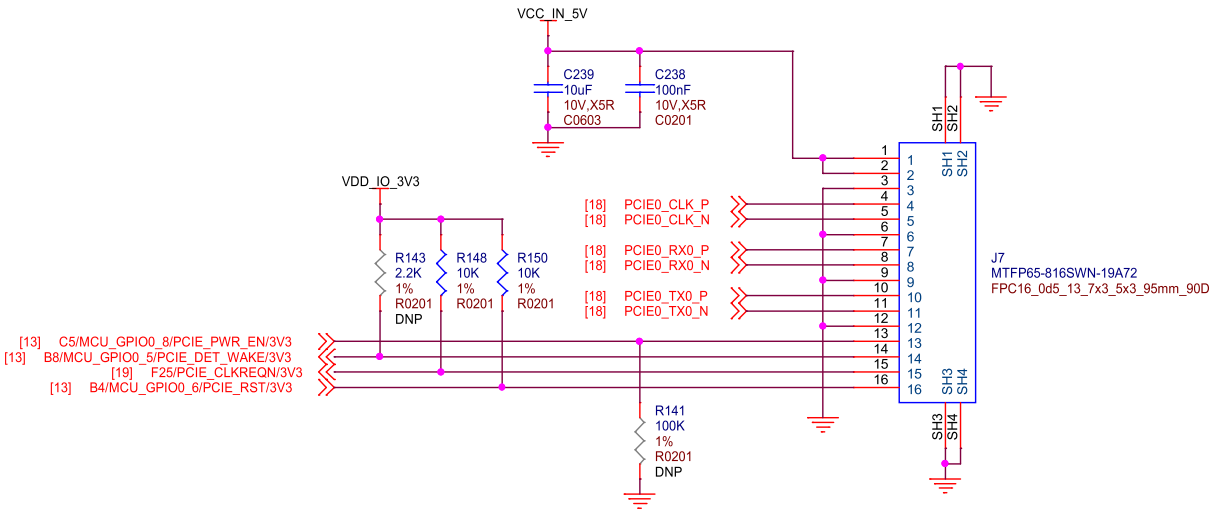


Fig. 3.33: BeagleY-AI PCIE connector

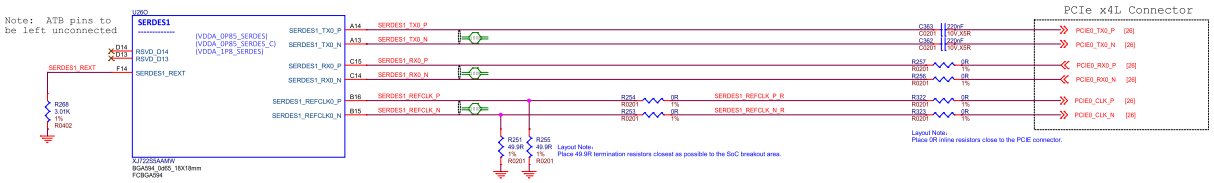


Fig. 3.34: BeagleY-AI SoC SERDES1

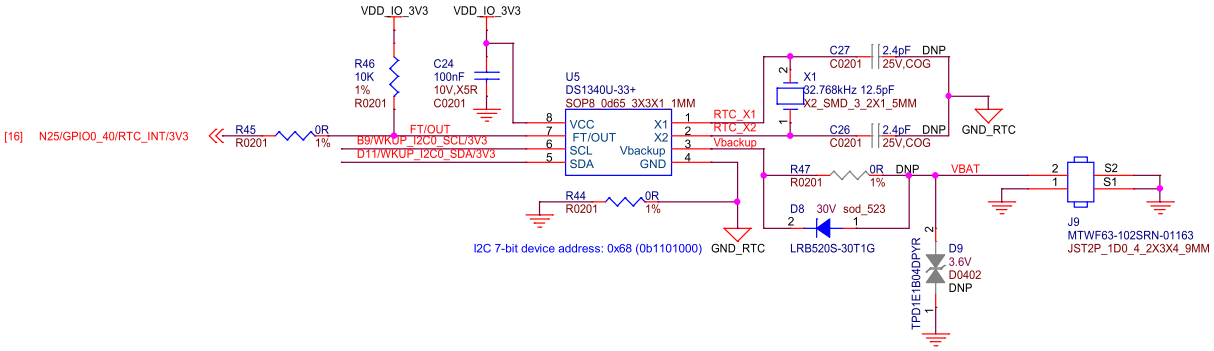


Fig. 3.35: BeagleY-AI I2C ext RTC

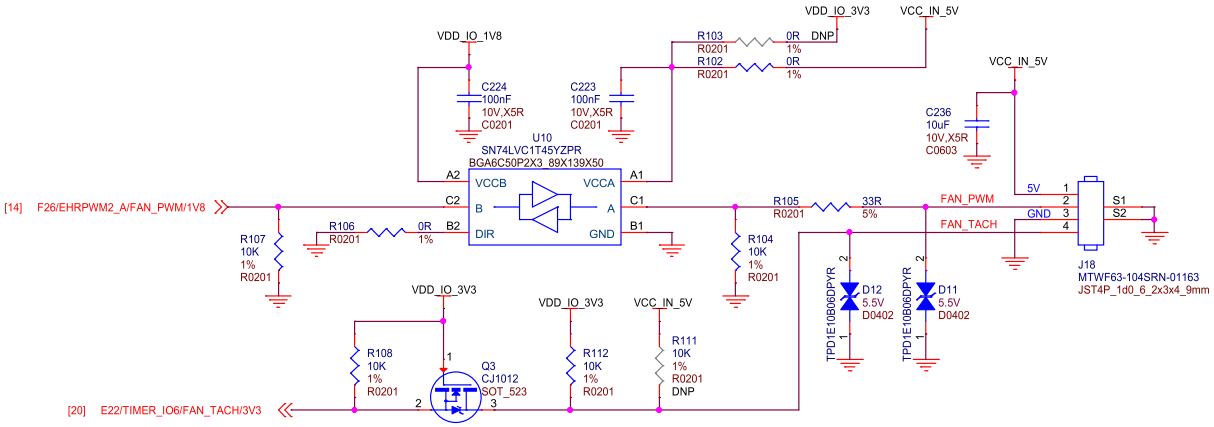


Fig. 3.36: BeagleY-AI fan connector

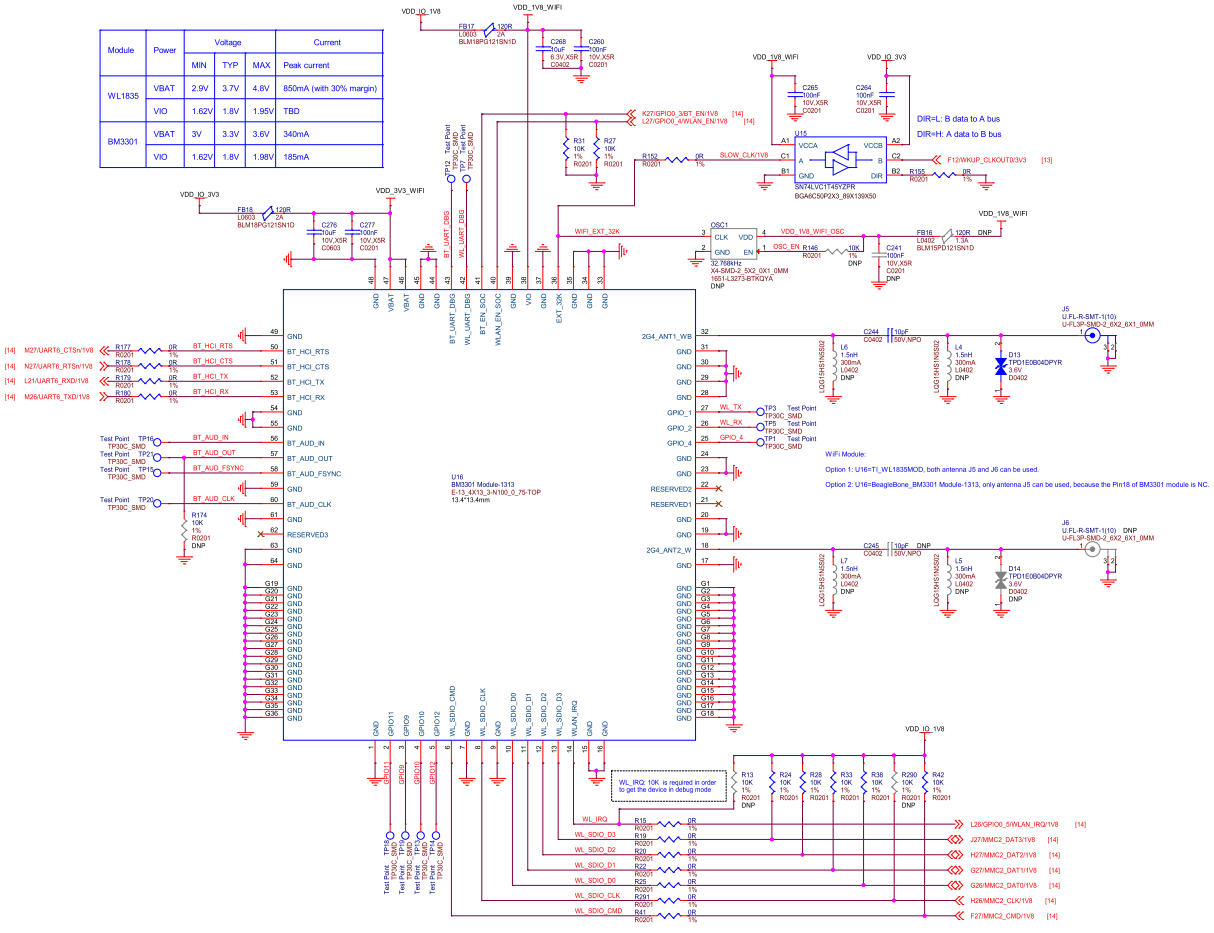


Fig. 3.37: BeagleY-AI WiFi module

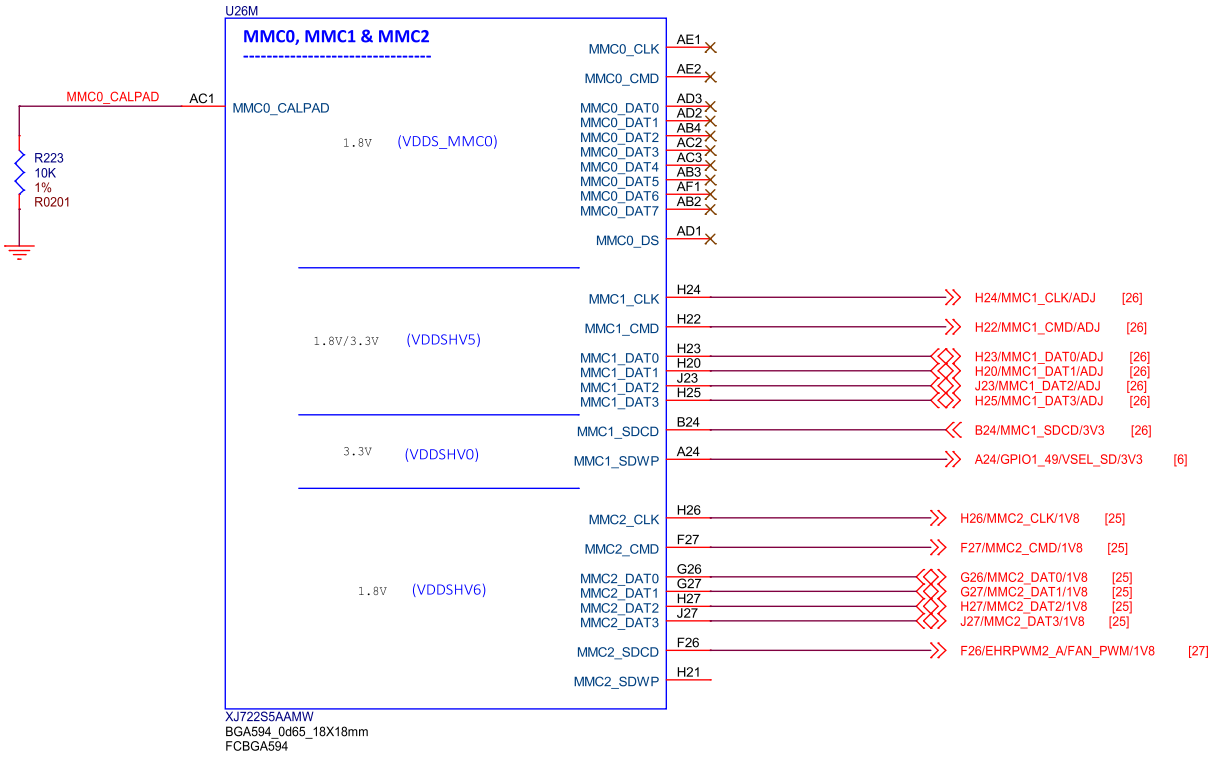


Fig. 3.38: BeagleY-AI SoC MMC0, MMC1, and MMC2

3.8.2 Ethernet

BeagleY-AI is equipped with a 1 Gb (10/100/1000) DP83867 Ethernet PHY connected over RGMII.

BeagleY-AI uses an RJ45 ethernet connector with integrated magnetics.

Optional PoE (Power over Ethernet) can also be used with compatible 3rd party HATs designed for the Raspberry Pi 5.

Note: Only Pi 5 PoE HATs are compatible, as Pi 4 and previous designs have the PoE pins in a different location.

3.9 Cameras & Displays

BeagleY-AI is capable of driving up to 3 Displays (HDMI, OLDI/LVDS & DSI) simultaneously.

- HDMI via DPI Converter up to 1920 x 1080 @60FPS
- OLDI/LVDS up to 3840 x 1080 @60FPS (Dual Link, 150-Mhz Pixel Clock)
- DSI up to 3840 x 1080 at 60fps (4 Lane MIPI® D-PHY, 300-MHz Pixel Clock)

It also features 2 CSI interfaces and can support up to 8 Cameras using Virtual Channels and V3Link.

Note: The CSI1/DSI0 22-pin port is muxed between the two interfaces like the RPi 5, meaning that you must chose if it's used as a Display or Camera port. The CSI0 22-pin connector can only be used as a Camera port.

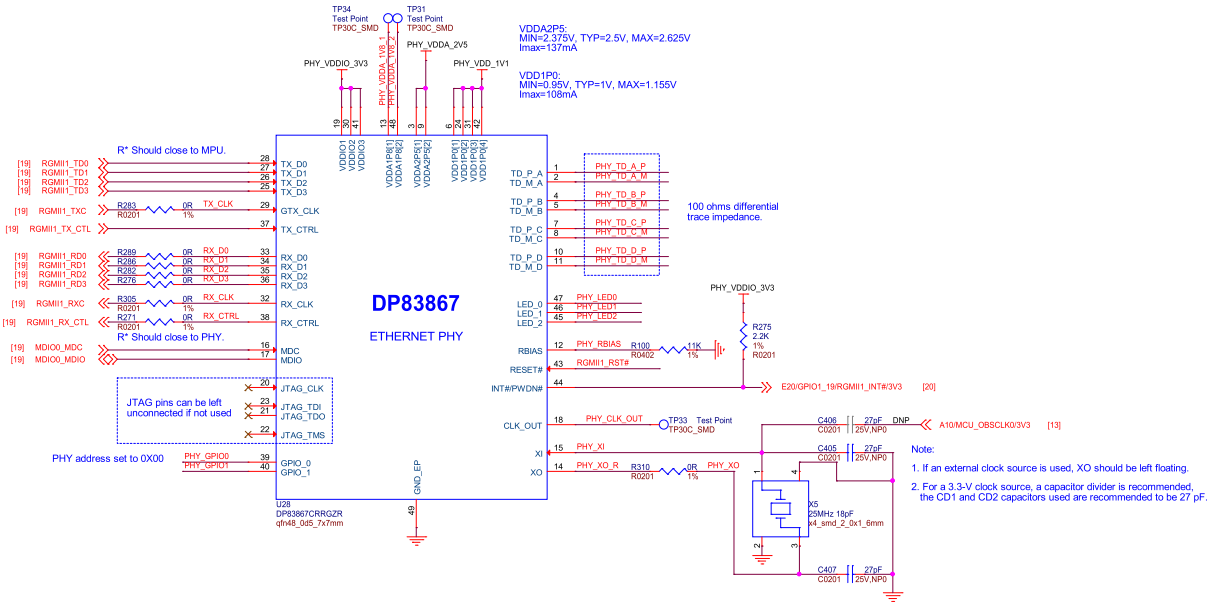


Fig. 3.39: BeagleY-AI ethernet DP83867

3.9.1 HDMI (DPI)

BeagleY-AI has a single HDMI 1.4 port capable of up to 1080p @60FPS with Audio. This is achieved using an external Parallel RGB (DPI) to HDMI converter from ITE.

Because the DPI interface is used up by the HDMI converter, it does mean that DPI is not available on the 40Pin GPIO header.

3.9.2 OLDI (LVDS)

The OLDI connector on BeagleY-AI has the same pinout as the one used by Beagle Play, meaning the same displays are compatible.

3.9.3 DSI

The DSI0 port is shared withe CSI1 and selectable via a MUX switch to maintain Pi functionality. It features the same pinout found on the 22-pin DSI connector on RPi5 and BeagleBone AI-64 and enables connectivity to existing supported DSI displays.

Please note that DSI is only available on the second of the two 22-pin “CSI” connectors.

3.9.4 CSI

To maintain a Pi compatible form factor, BeagleY-AI only exposes 2 of the 4 physical CSI interfaces of the AM67A SoC. Each CSI interfaces is MIPI® CSI-2 v1.3 + MIPI® D-PHY 1.2 with 4 Data Lanes running at up to 2.5Gbps/lane. The interface also supports up to 16 Virtual Channels for multi-camera applications using FPDLink or V3Link.

3.10 Buttons and LEDs

BeagleY-AI features a single dual-color (Red/Green) LED for Power/Status indication.

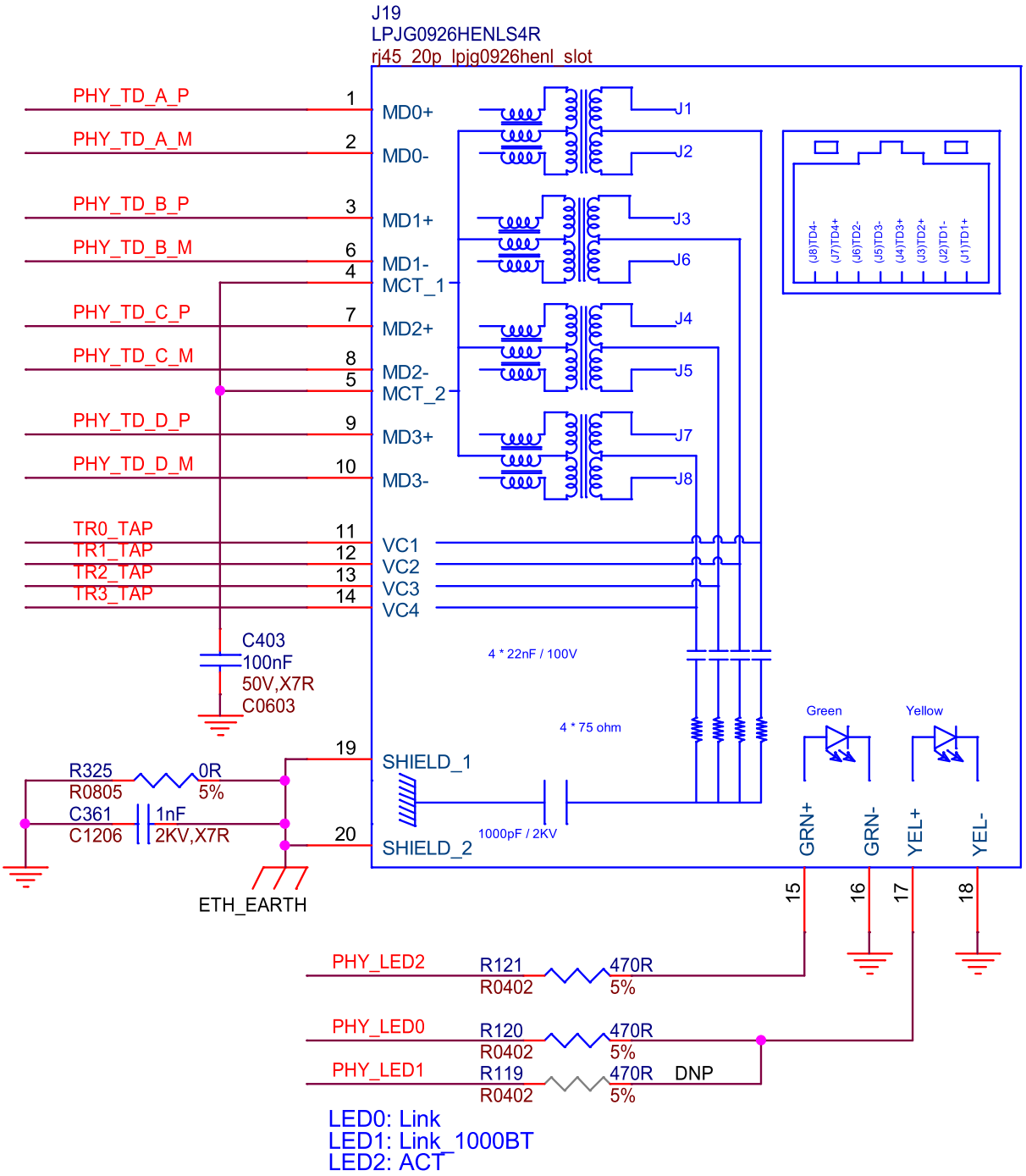


Fig. 3.40: BeagleY-AI ethernet connector

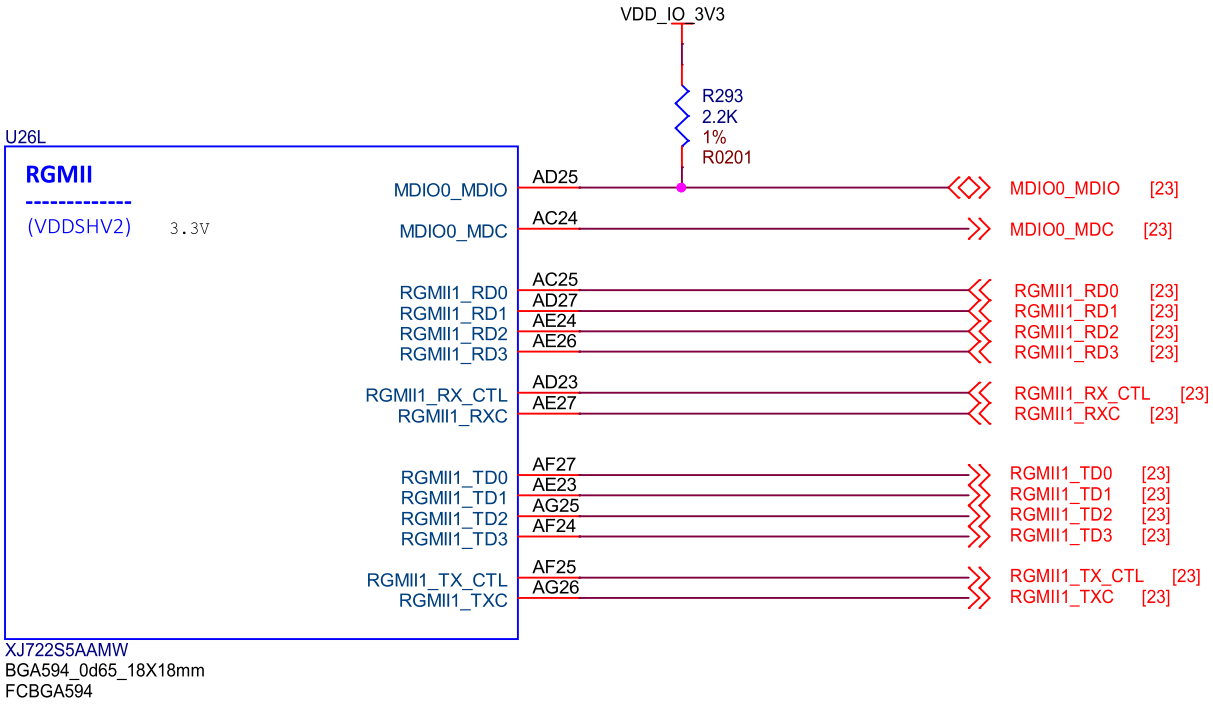


Fig. 3.41: BeagleY-AI SoC RGMII

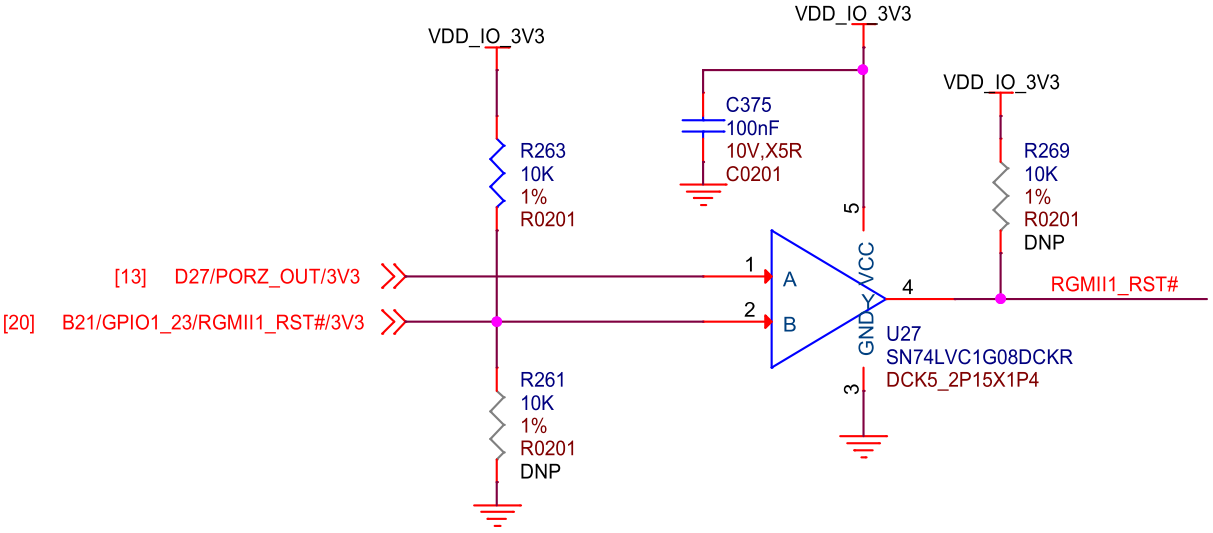


Fig. 3.42: BeagleY-AI SoC RGMII1 RST

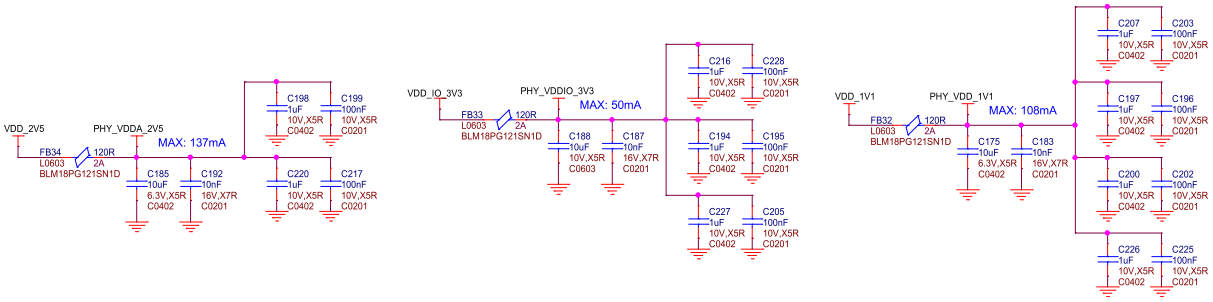


Fig. 3.43: BeagleY-AI Ethernet PHY caps

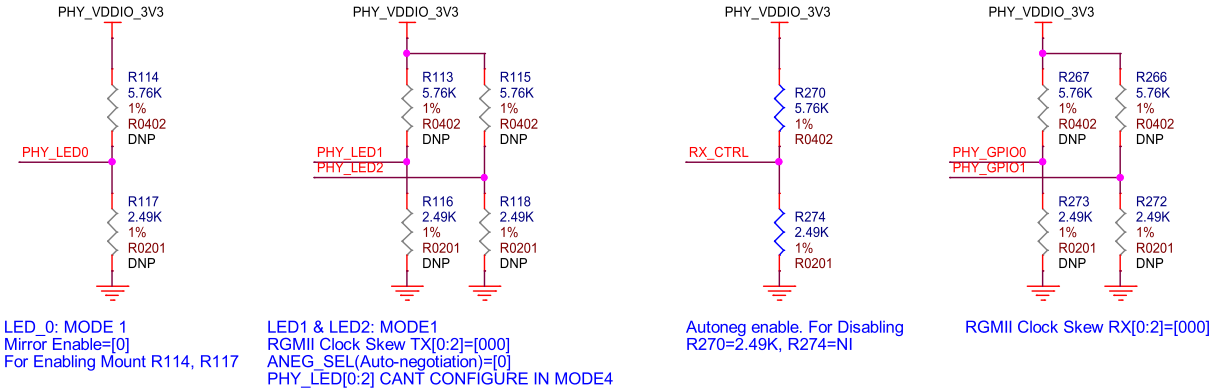


Fig. 3.44: BeagleY-AI Ethernet PHY misc

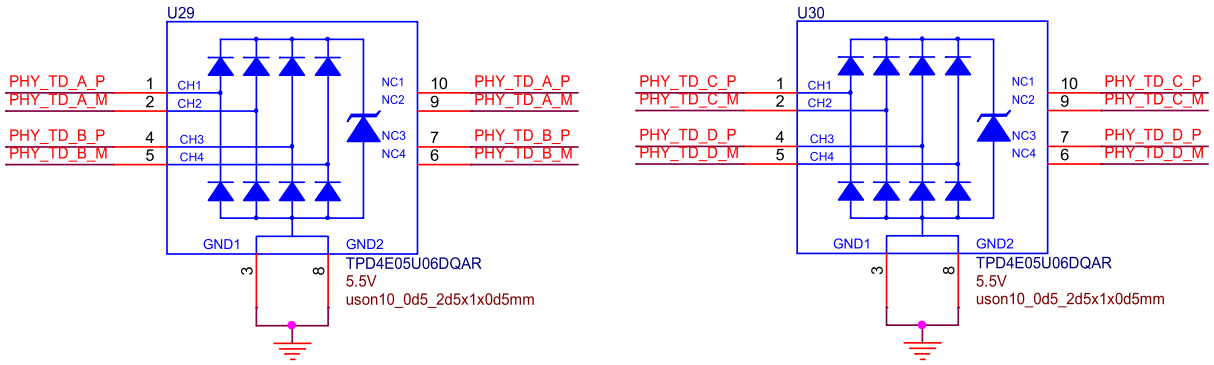


Fig. 3.45: BeagleY-AI Ethernet PHY protection

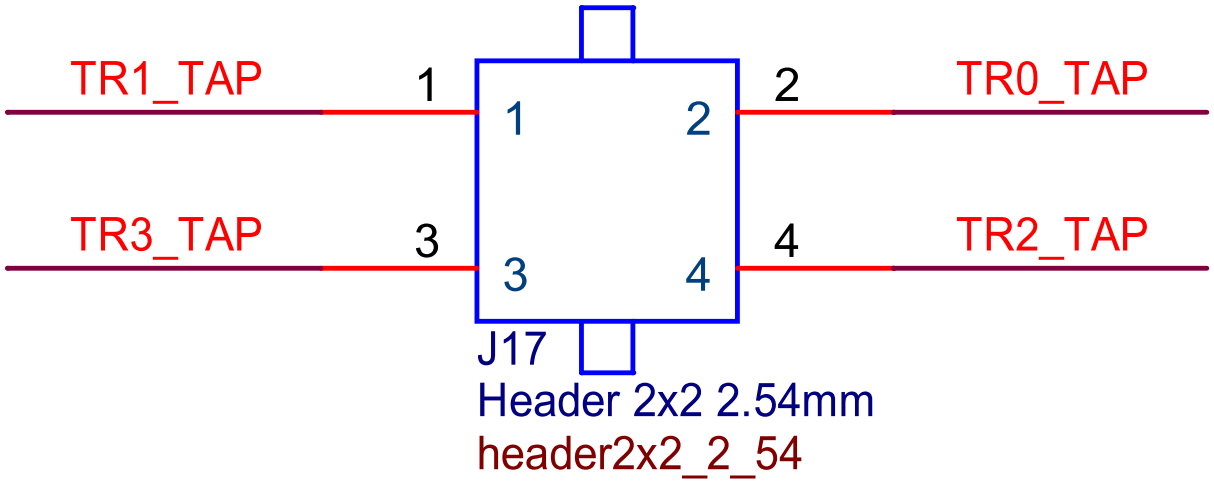


Fig. 3.46: BeagleY-AI PoE header

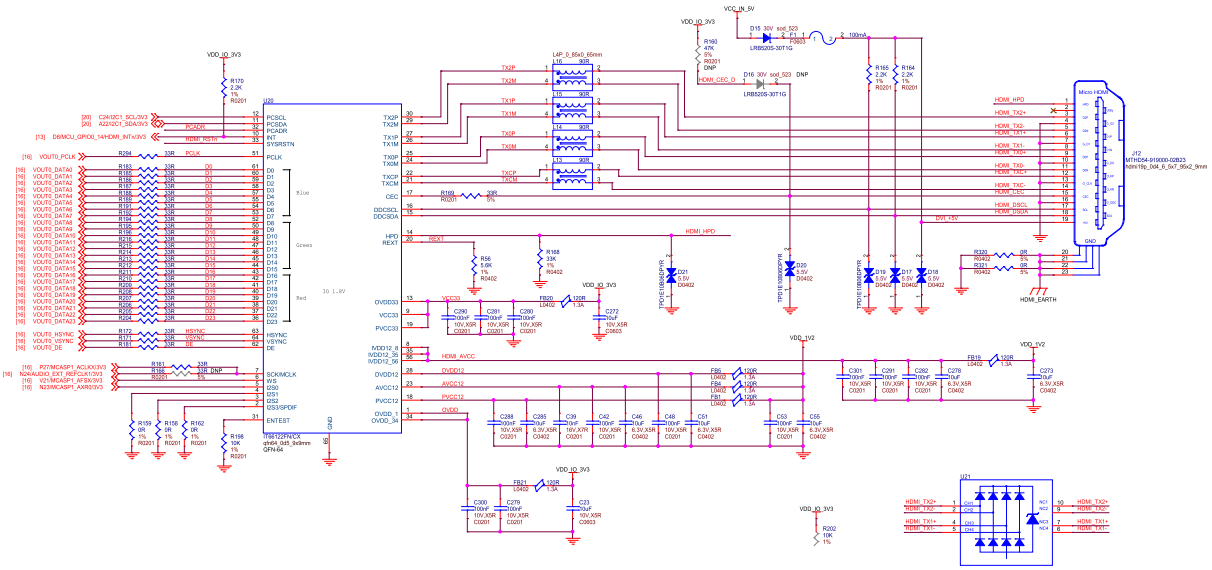


Fig. 3.47: BeagleY-AI RGB888 to HDMI

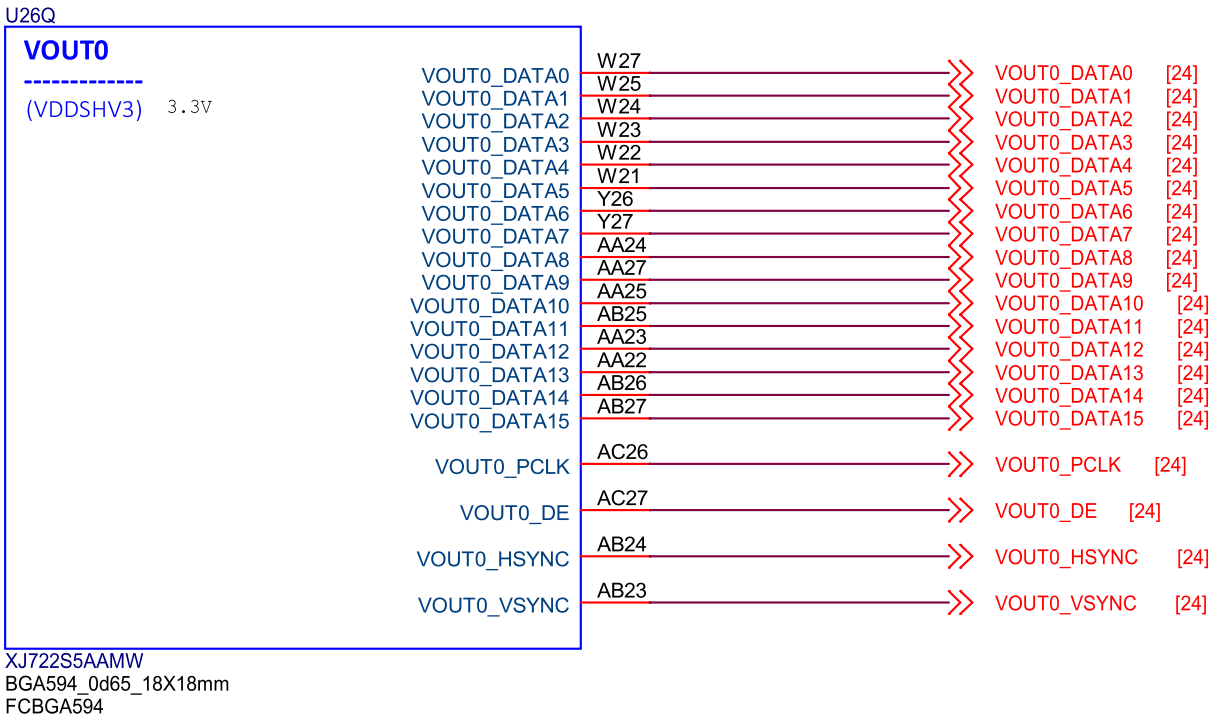


Fig. 3.48: BeagleY-AI SoC VOUT

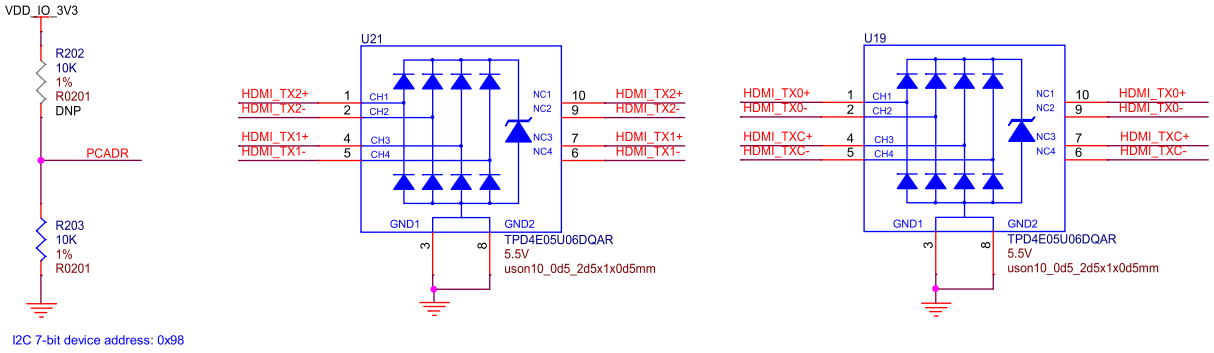


Fig. 3.49: BeagleY-AI HDMI addr protection

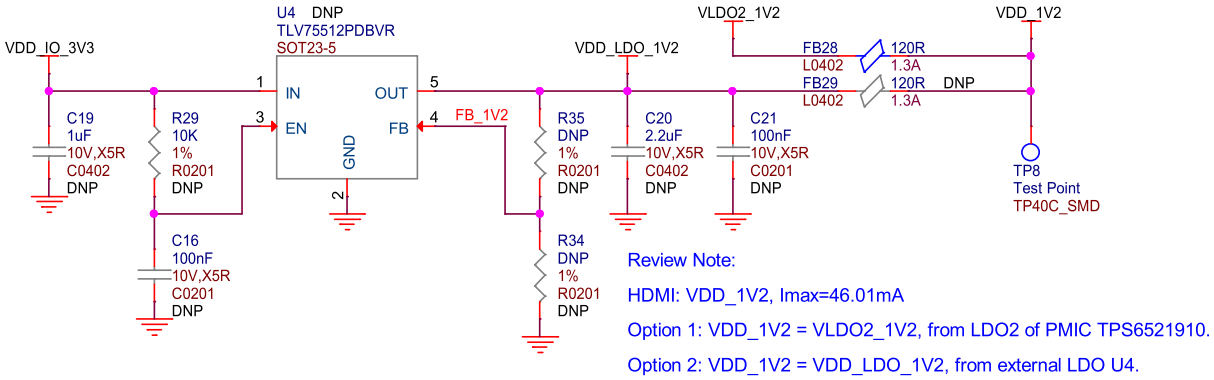


Fig. 3.50: BeagleY-AI HDMI power

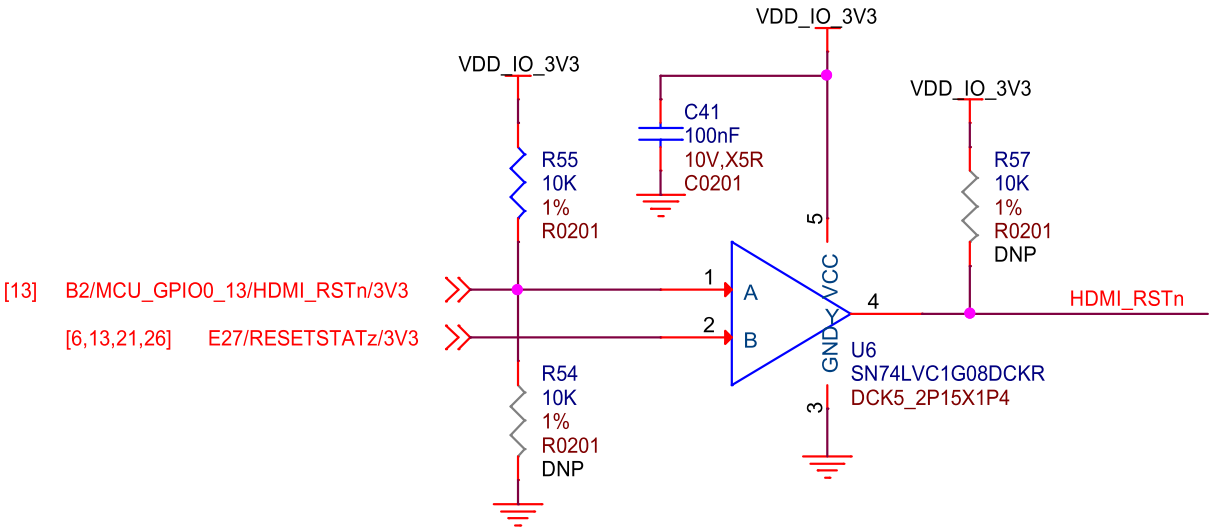


Fig. 3.51: BeagleY-AI HDMI reset

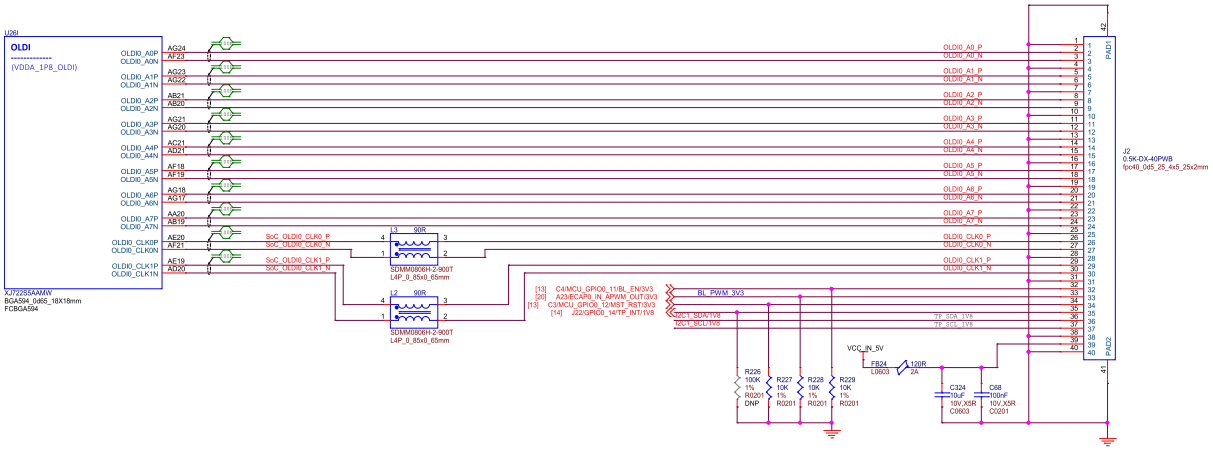


Fig. 3.52: BeagleY-AI SoC OLDI

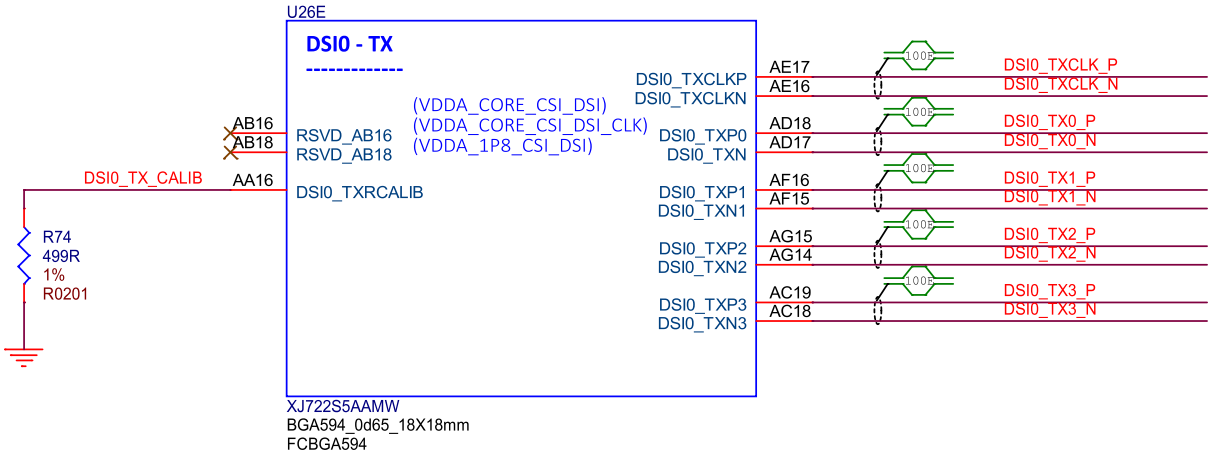


Fig. 3.53: BeagleY-AI SoC DSI0 TX connections

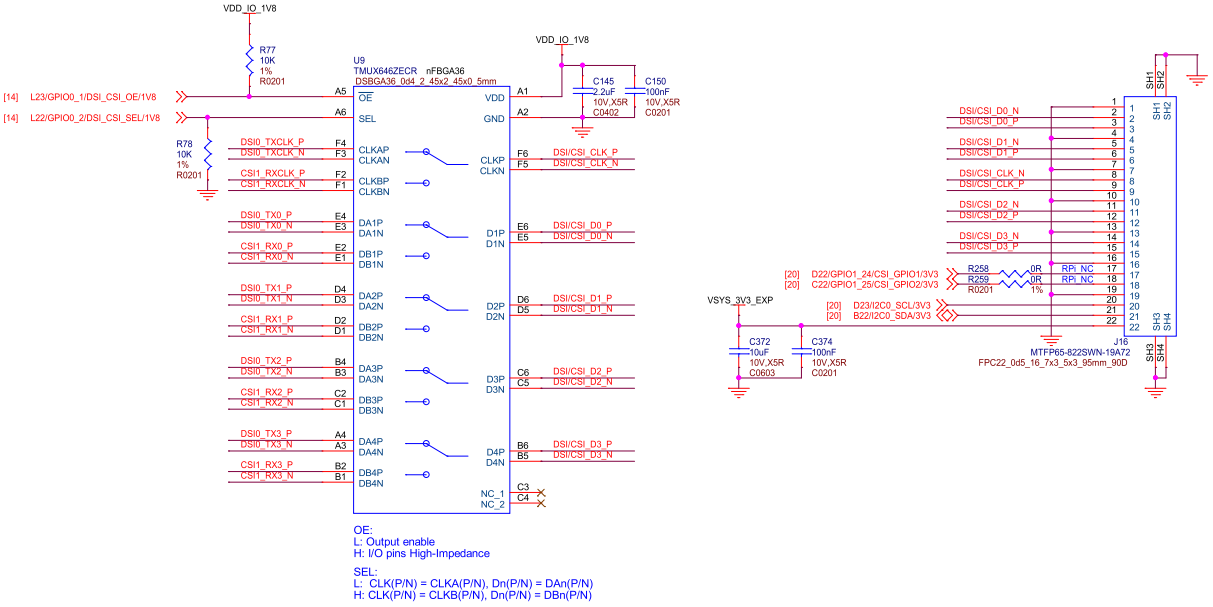


Fig. 3.54: BeagleY-AI RPI DSI/CSI

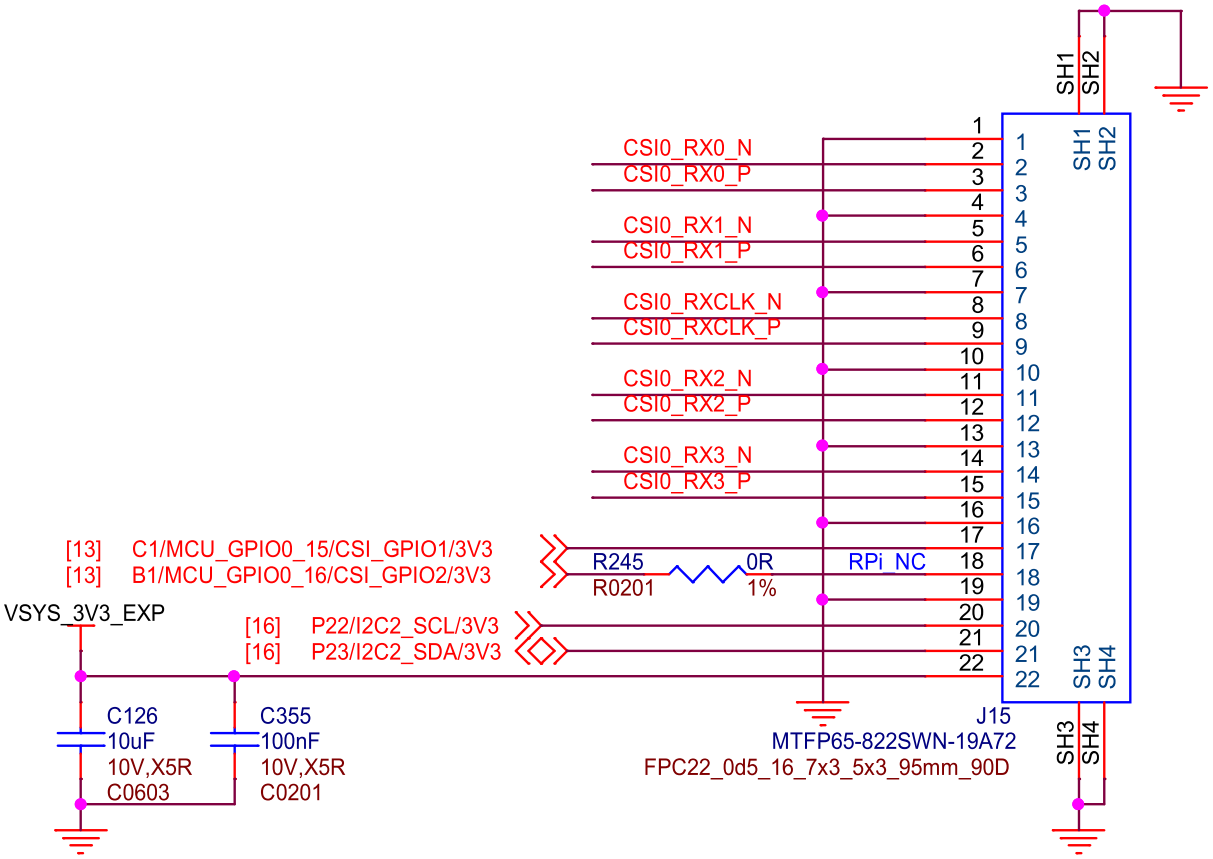


Fig. 3.55: BeagleY-AI RPI CSI

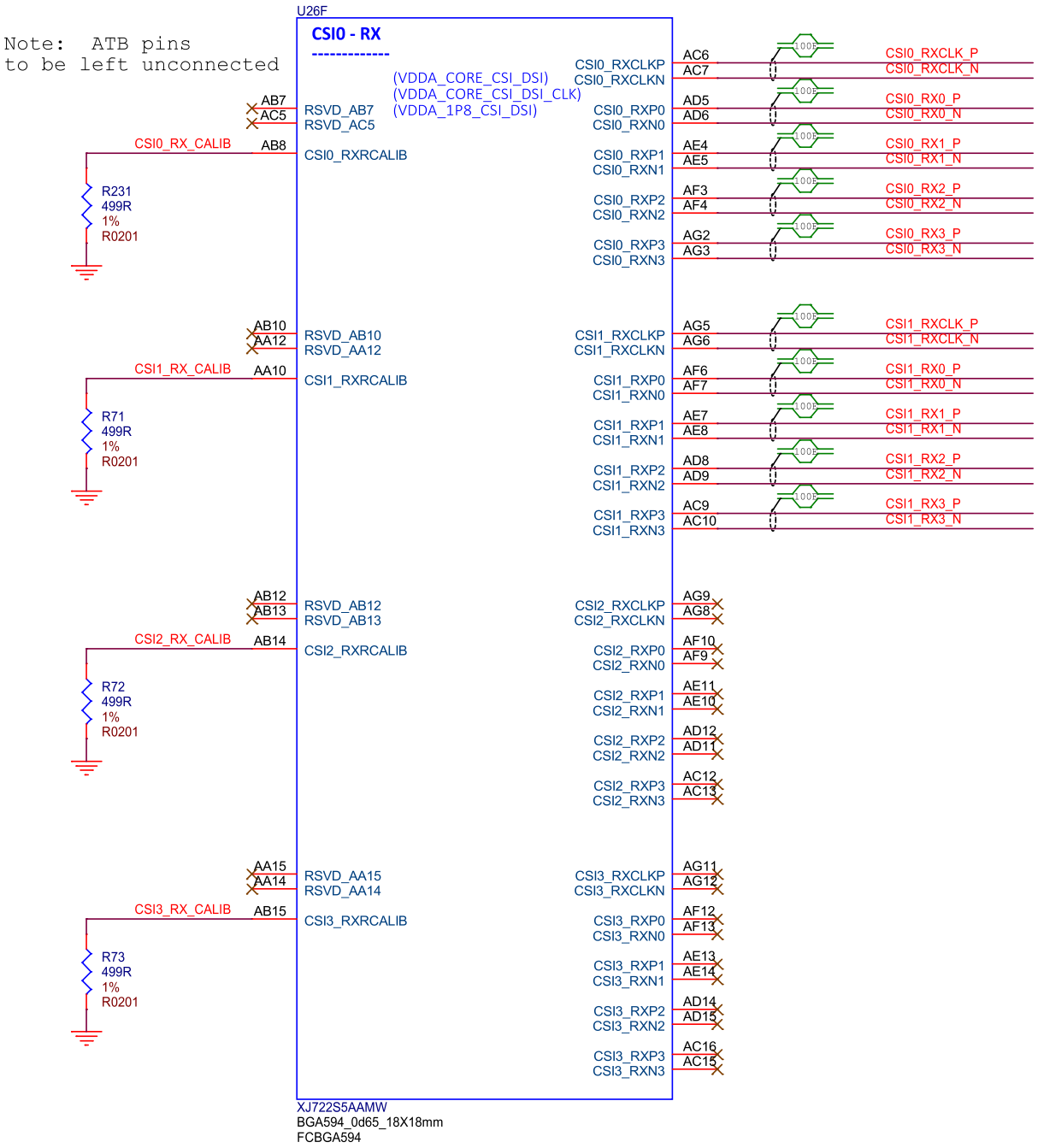


Fig. 3.56: BeagleY-AI SoC CSI1, CSI2, and CSI3

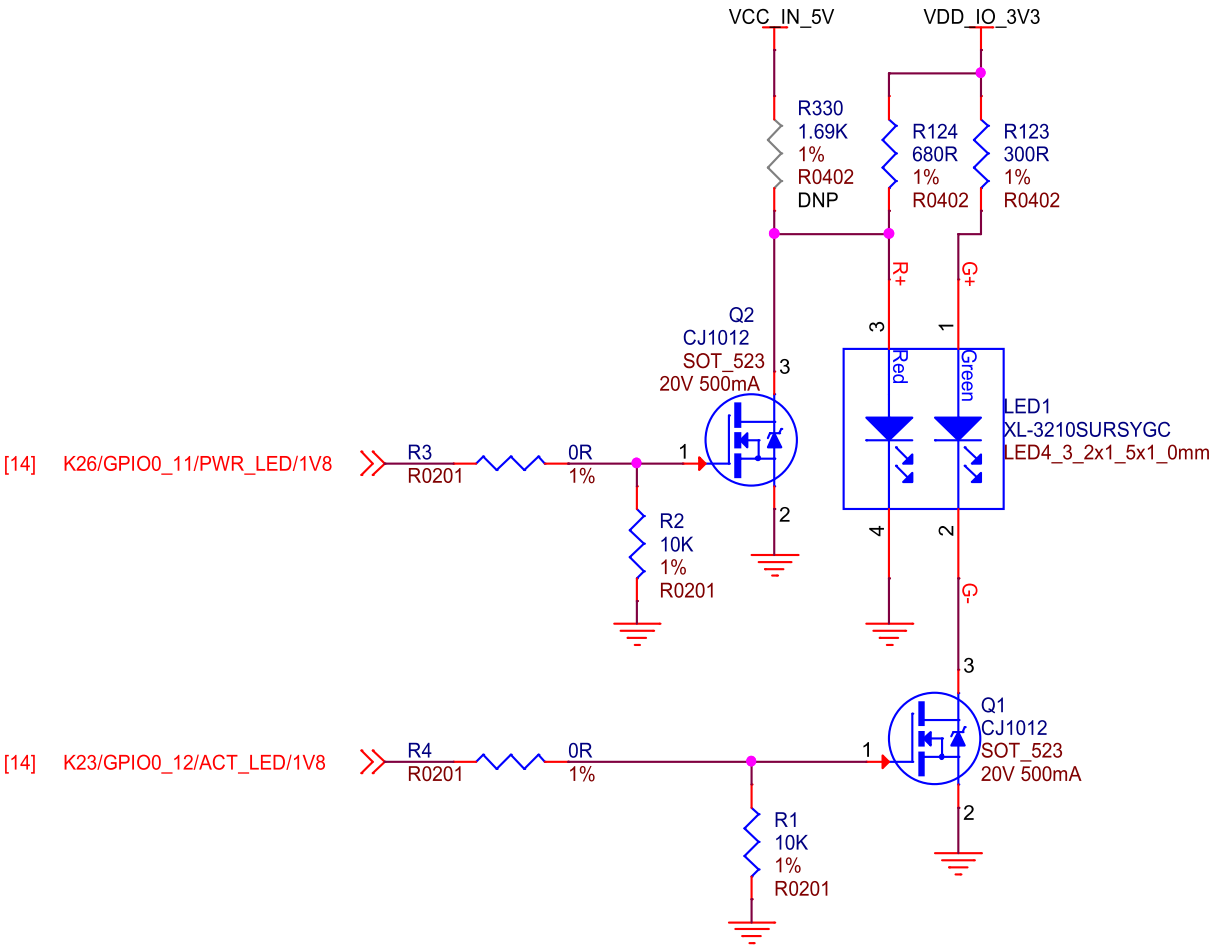


Fig. 3.57: BeagleY-AI LEDs

3.11 Debug Ports

3.11.1 JTAG Tag-Connect

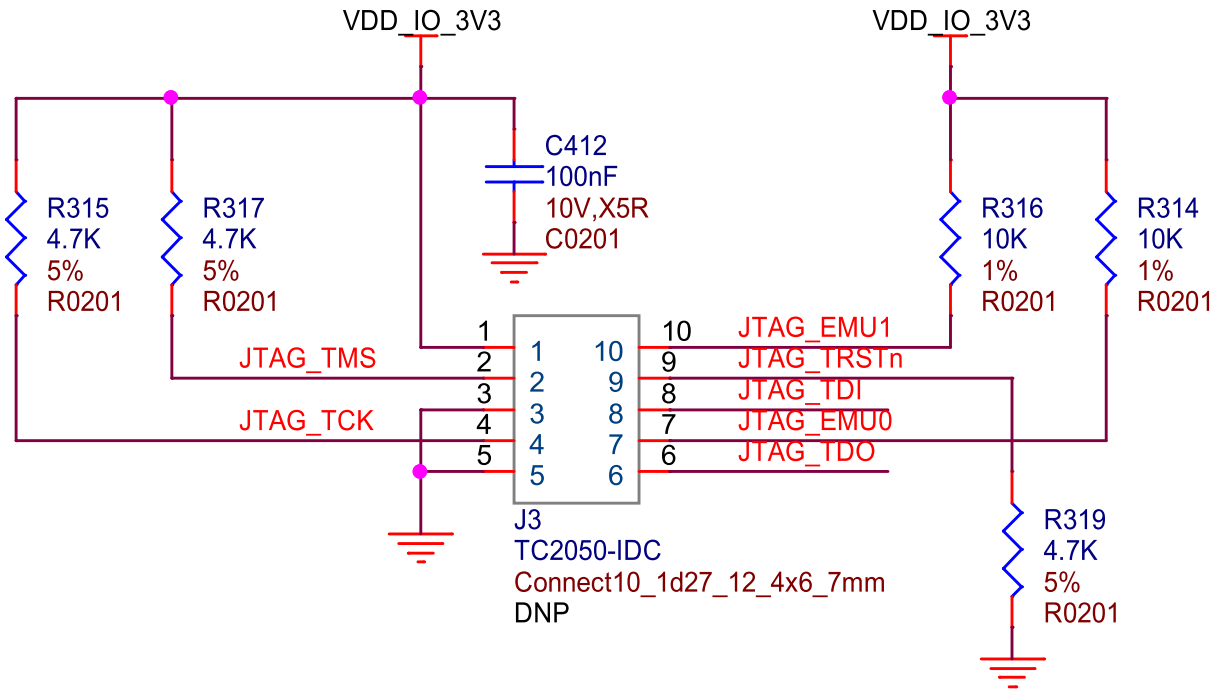


Fig. 3.58: BeagleY-AI Tag-Connect

JTAG is available on the BeagleY-AI via a 10pin Tag-Connect header located on the bottom of the board between the USB 3.0 ports.

Because of the density of the board and tight fit of the USB connectors, the standard retention clip provided by Tag-Connect will not fit. A recommended 3D printable adapter is available on Printables

3.11.2 UART

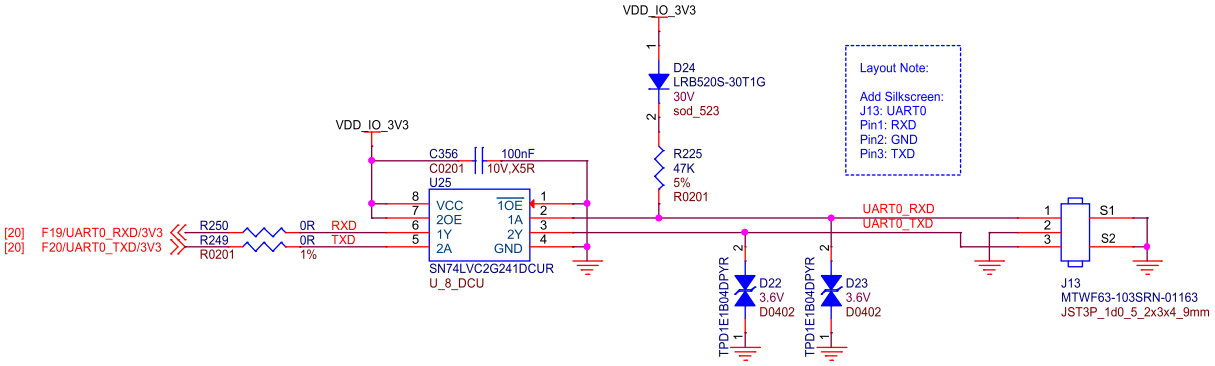


Fig. 3.59: BeagleY-AI debug UART port

By default, BeagleY-AI exposes the UART port used by UBoot & Linux on a Pi Debugger compatible JST 3pin header. The UART port used for debug can also be changed in software to use a UART available on the 40Pin GPIO header.

3.11.3 PMIC NVM Tag-Connect

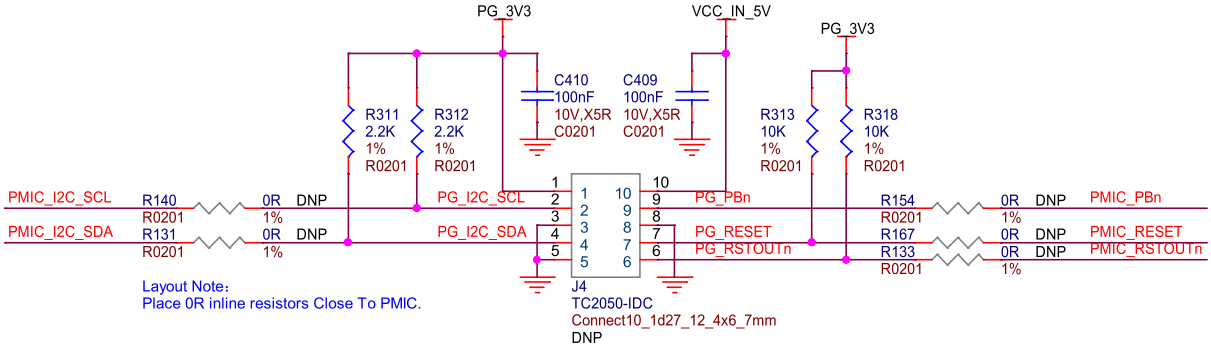


Fig. 3.60: BeagleY-AI PMIC NVM programming interface

A PMIC programming header is present on the BeagleY-AI in the form of a 10pin Tag-Connect header located on the bottom of the board between the Ethernet and USB 3.0 ports. Ensure you do not connect JTAG to this port as the pinout and interface is different. PMIC NVM programming should not be performed unless you know what you're doing. The port is mainly intended for use during manufacturing.

3.12 Miscellaneous

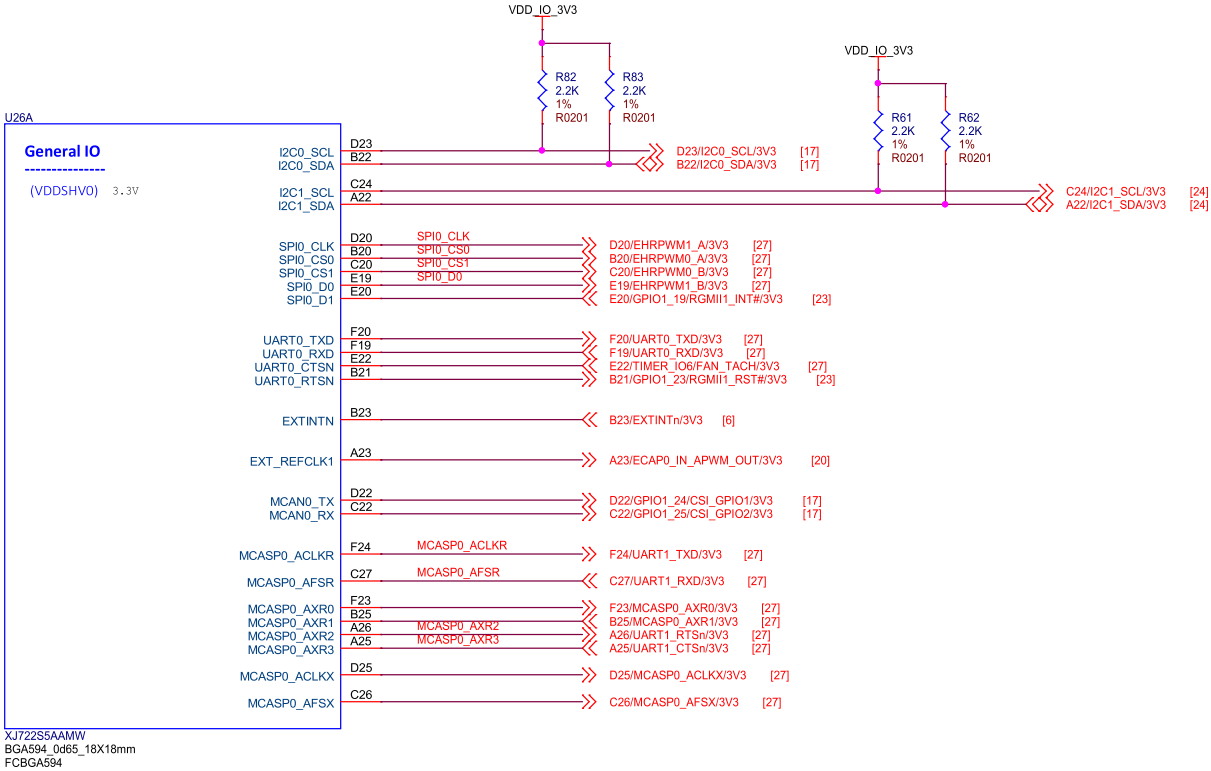


Fig. 3.61: BeagleY-AI general IO

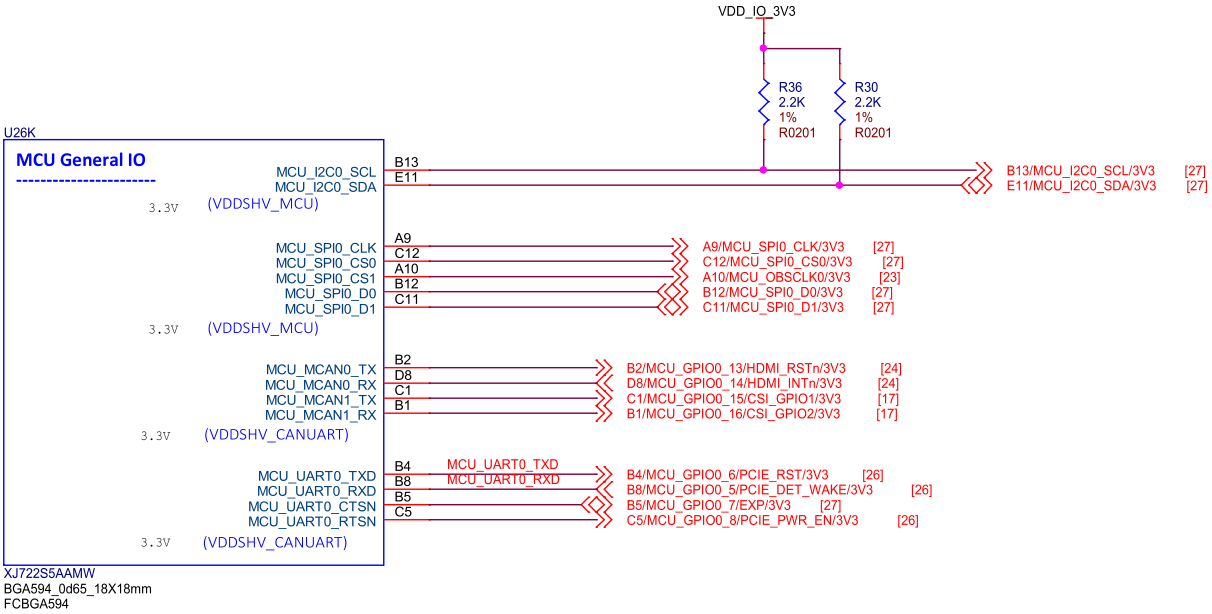


Fig. 3.62: BeagleY-AI MCU general IO

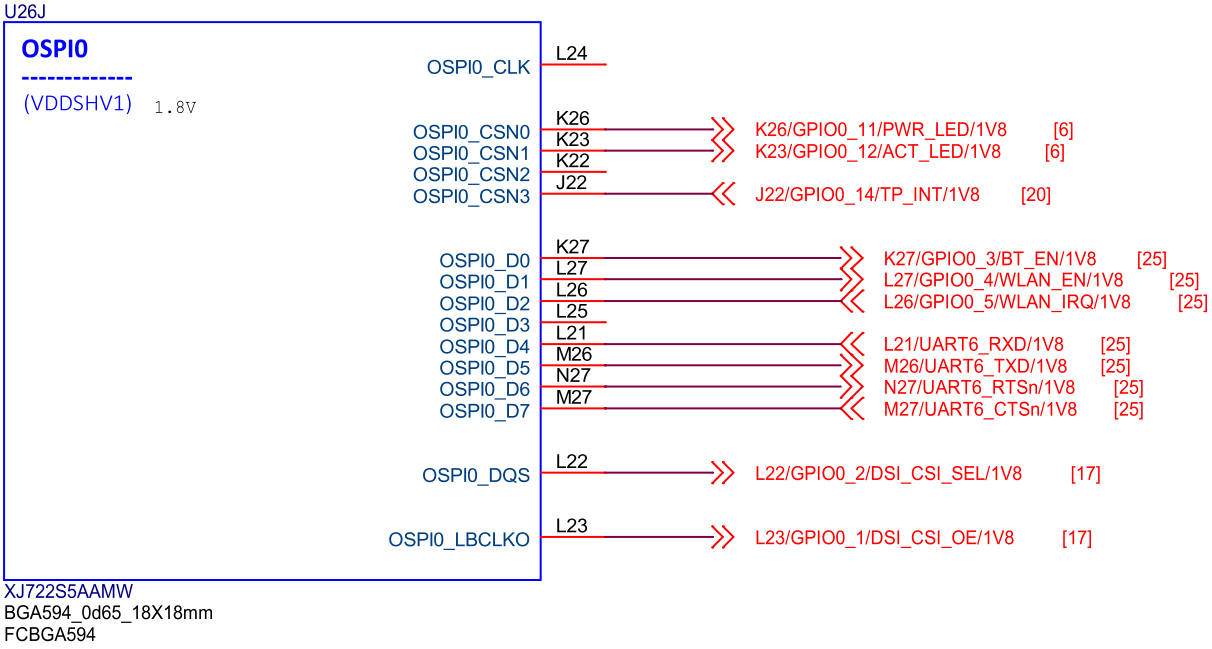


Fig. 3.63: BeagleY-AI SoC OSPI0

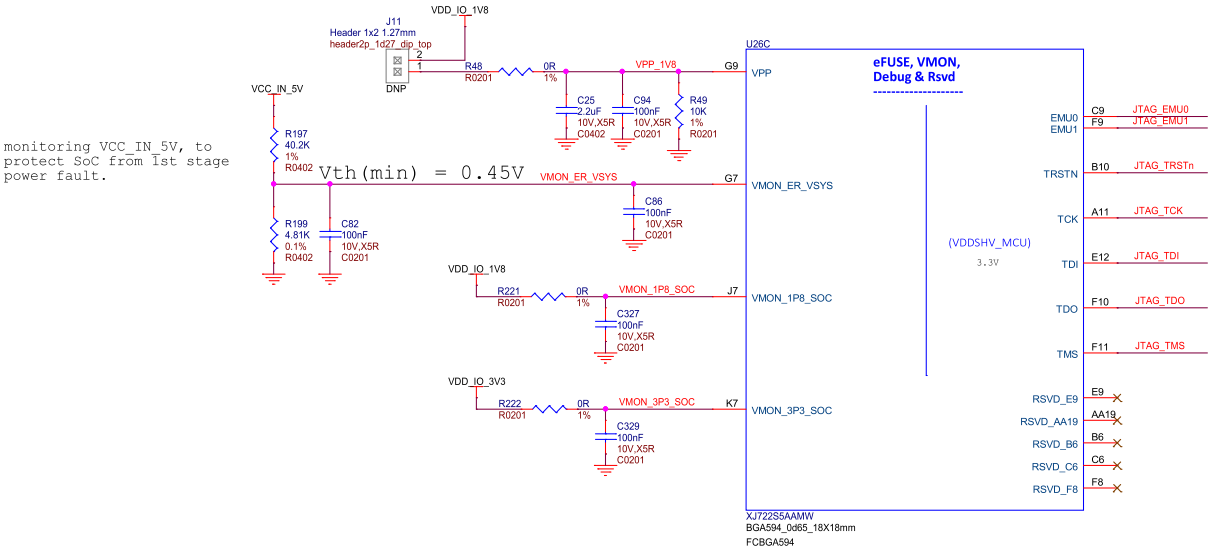


Fig. 3.64: BeagleY-AI SoC eFUSE, VMON, Debug, and RSVD

3.13 Mechanical Specifications

Table 3.1: Dimensions & weight

Parameter	Value
Size	85 x 56 x 20 mm
Max heigh	20mm
PCB Size	85 x 56 mm
PCB Layers	14 layers
PCB Thickness	1.6mm
RoHS compliant	Yes
Gross Weight	110 g
Net Weight	50 g

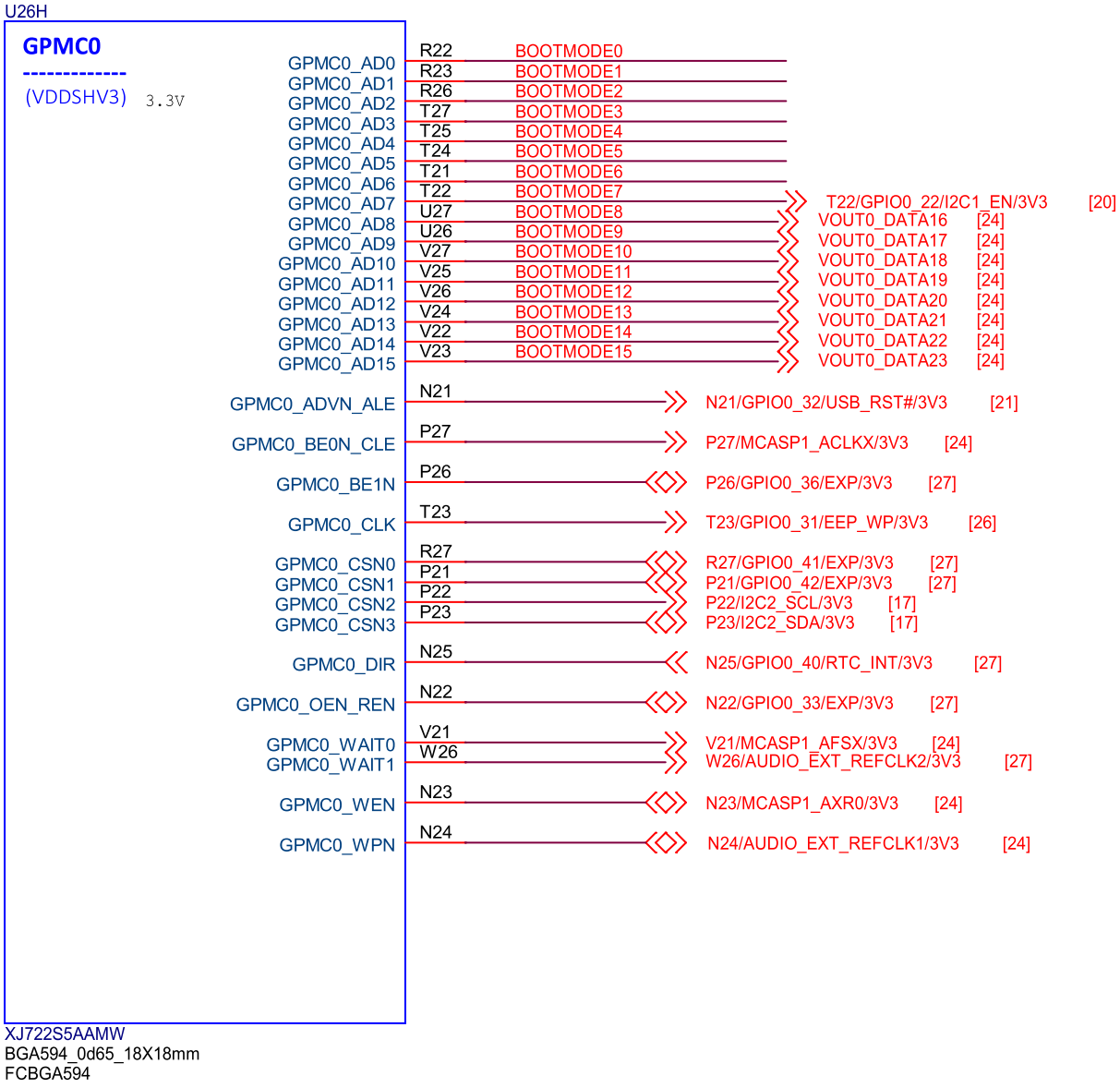


Fig. 3.65: BeagleY-AI SoC GPMC0

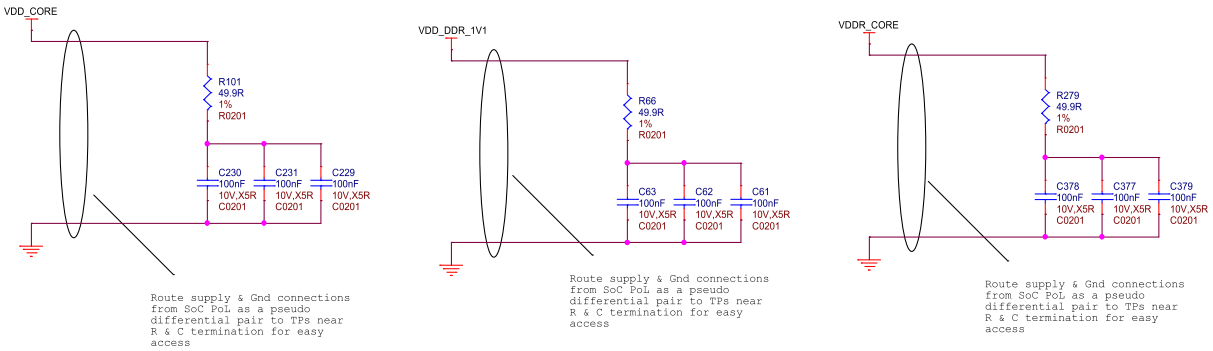


Fig. 3.66: BeagleY-AI SoC supply noise kelvin sensing

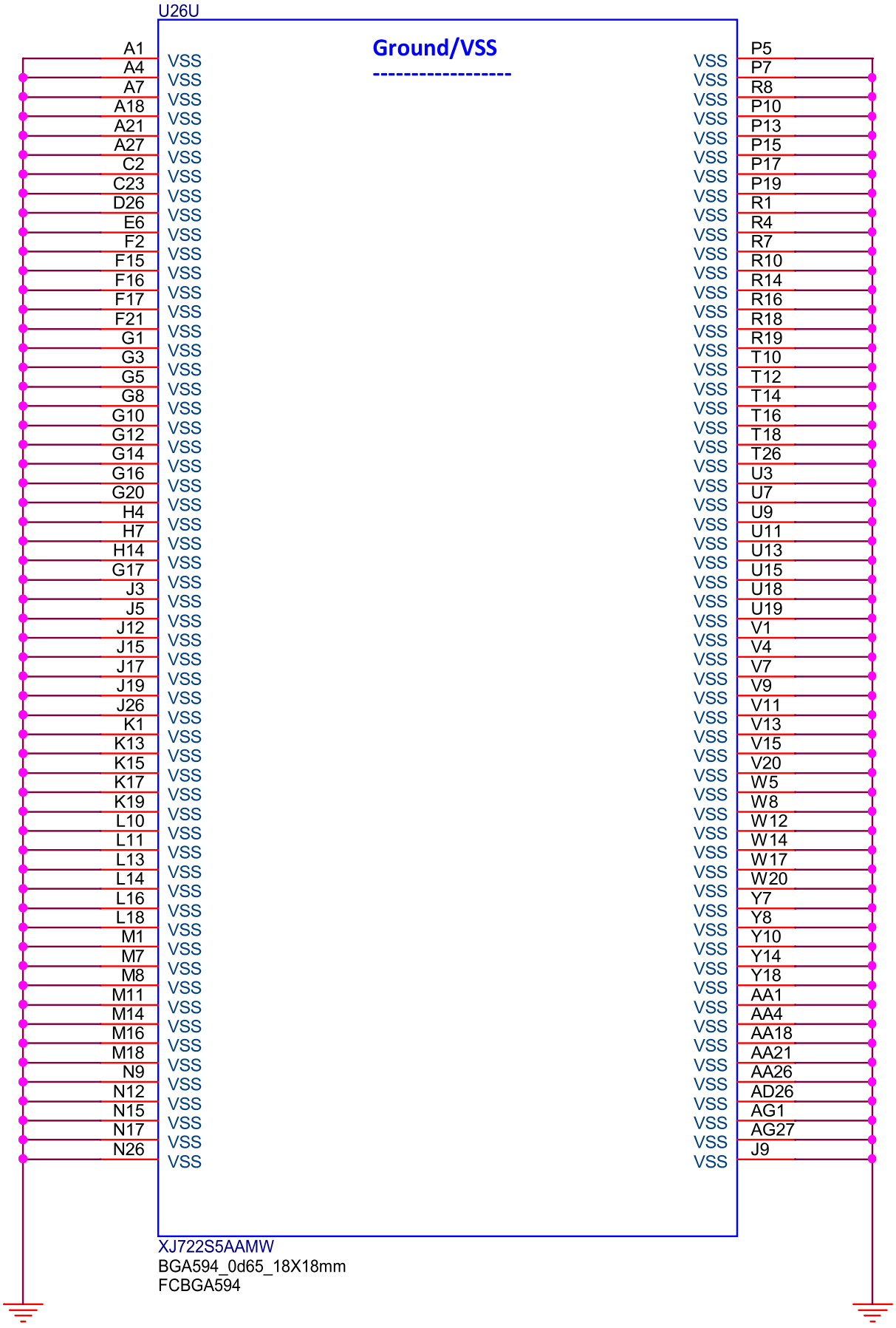


Fig. 3.67: BeagleY-AI SoC ground connections

Chapter 4

Expansion

Todo: Describe how to build expansion hardware for BeagleY-AI

4.1 PCIe

For software reference, you can see how PCIe is used on NVMe HATs.

- *Booting from NVMe Drives*
- *Using IMX219 CSI Cameras*
- *Using the on-board Real Time Clock (RTC)*

Chapter 5

Demos and tutorials

5.1 Using GPIO

Todo: Add information about software image used for this demo.

GPIO stands for **General-Purpose Input/Output**. It's a set of programmable pins that you can use to connect and control various electronic components.

You can set each pin to either **read signals (input)** from things like buttons and sensors or **send signals (output)** to things like LEDs and motors. This lets you interact with and control the physical world using code!

A great resource for understanding pin numbering can be found at pinout.beagley.ai

Warning: BeagleY-AI GPIOs are 3.3V tolerant, using higher voltages **WILL DAMAGE** the processor!

5.1.1 Pin Numbering

You will see pins referenced in several ways. While this is confusing at first, in reality, we can pick our favorite way and stick to it.

The two main ways of referring to GPIOs is **by their number**, so GPIO 2, 3, 4 etc. as seen in the diagram below. This corresponds to the SoC naming convention. For broad compatibility, BeagleY-AI re-uses the Broadcom GPIO numbering scheme used by RaspberryPi.

The second (and arguably easier) way we will use for this tutorial is to use the **actual pin header number** (shown in dark grey)

So, for the rest of the tutorial, if we refer to **hat-08-gpio** we mean the **8th pin of the GPIO header**. Which, if you referenced the image below, can see refers to **GPIO 14 (UART TX)**

If you are curious about the "real" GPIO numbers on the Texas Instruments AM67A SoC, you can look at the board schematics.

5.1.2 Required Hardware

For the simple blink demo, all that is needed is an LED, a Resistor (we use 2.2K here) and 2 wires.

Similarly, a button is used for the GPIO read example, but you can also just connect that pin to 3.3V or GND with a wire to simulate a button press.

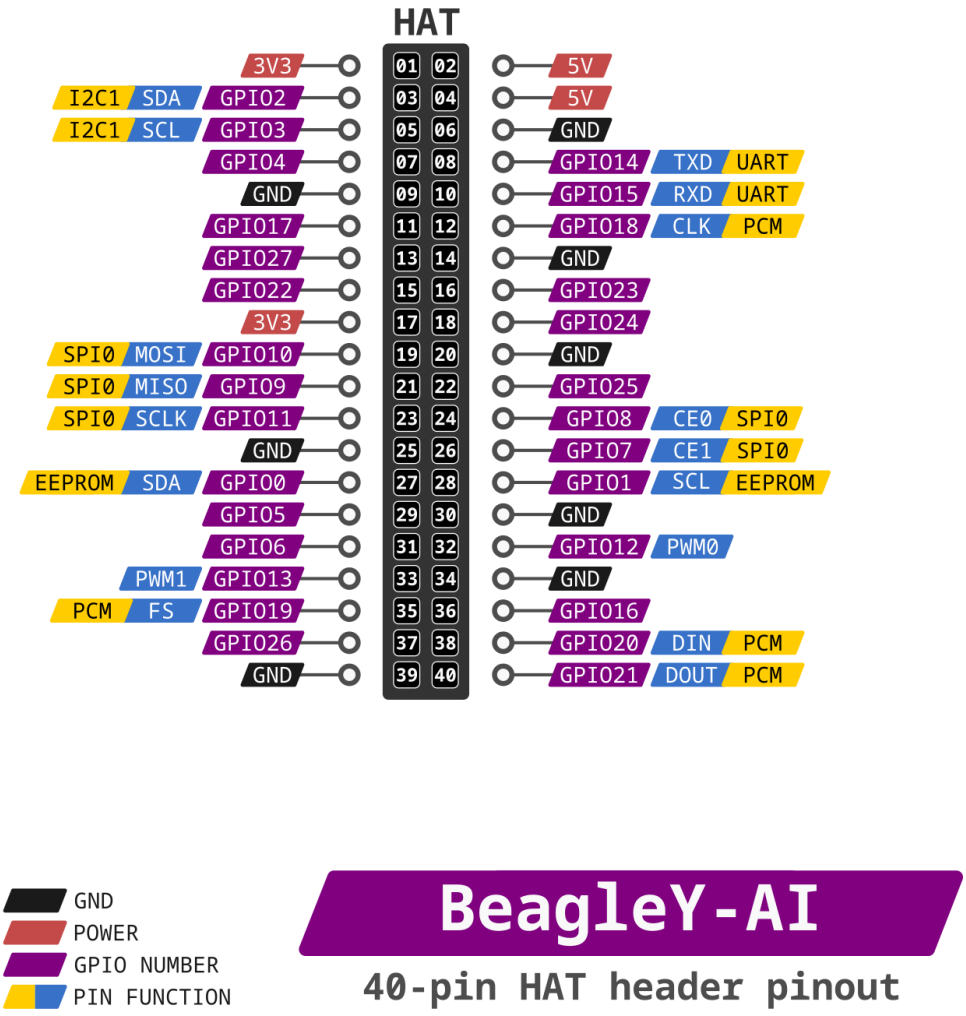


Fig. 5.1: BeagleY-AI pinout

Todo: Add fritzing diagram and chapter on Pin Binding here

5.1.3 GPIO Write

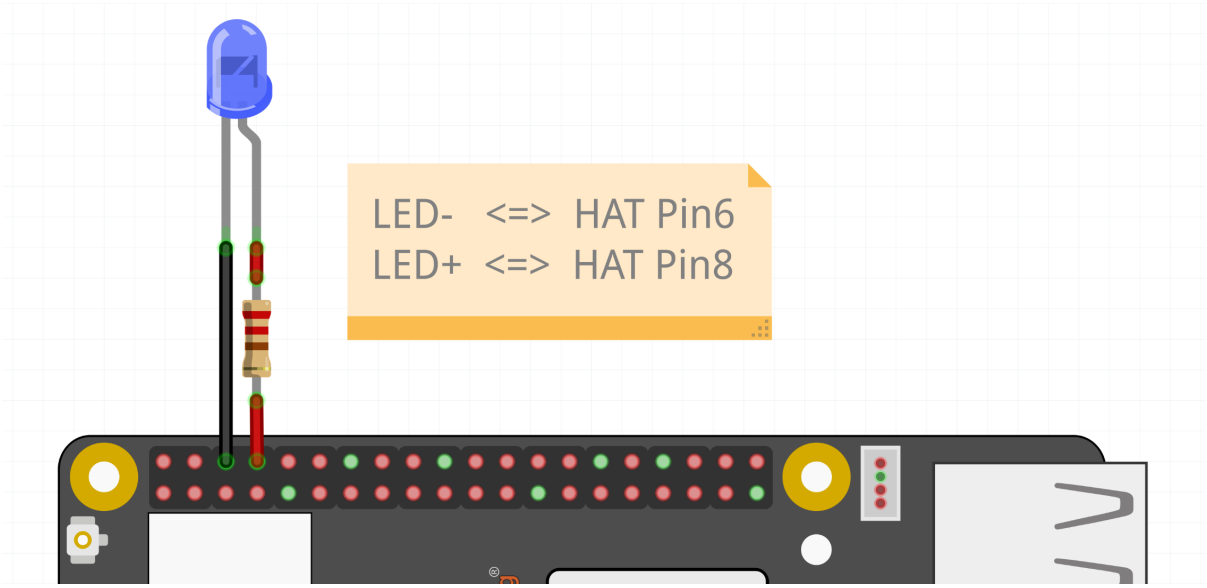


Fig. 5.2: LED connected to HAT Pin8 (GPIO14)

At it's most basic, we can set a GPIO using the **gpioset** command.

- To set HAT **Pin 8** (GPIO14) to **ON**:

```
gpioset $(gpiofind GPIO14)=1
```

- To set HAT **Pin 8** (GPIO14) to **OFF**:

```
gpioset $(gpiofind GPIO14)=0
```

5.1.4 Blink an LED

Let's create a script called **blinky.sh**,

- Create the file,

```
touch blinky.sh
```

- Open the file using `nano` editor,

```
nano blinky.sh
```

- Copy paste the code below to `blinky.sh` file,

```
#!/bin/bash

while :
do
      gpioset $(gpiofind GPIO14)=1
      sleep 1
```

(continues on next page)

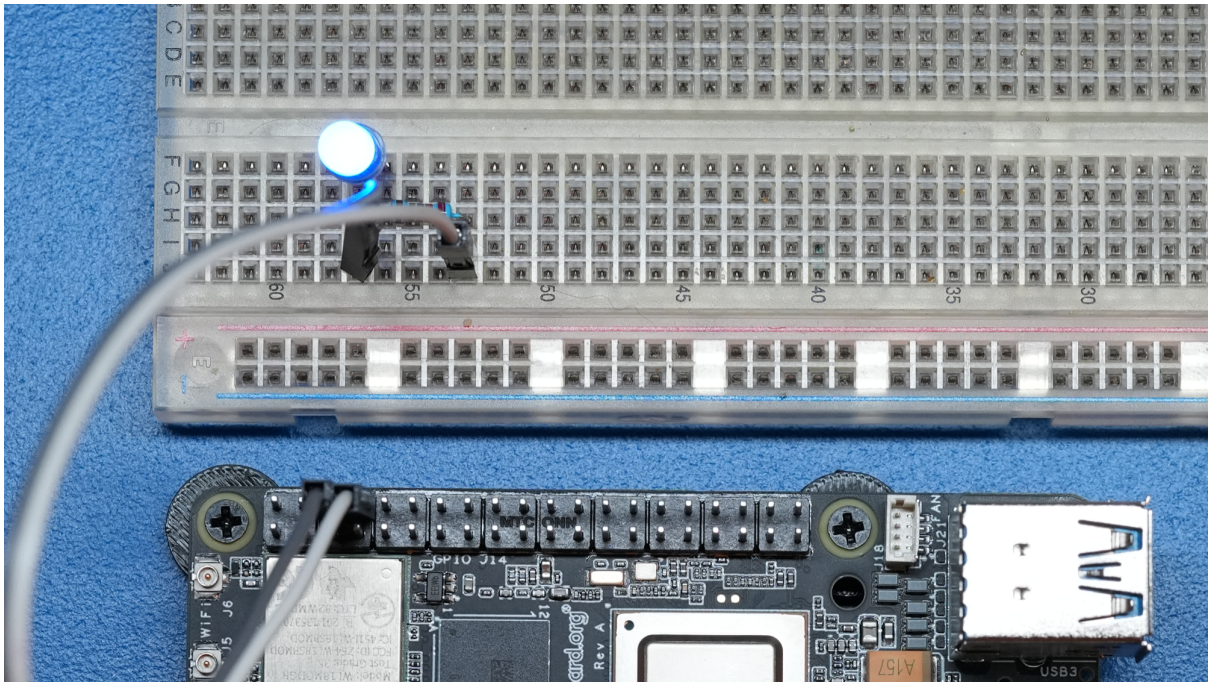


Fig. 5.3: GPIO ON state

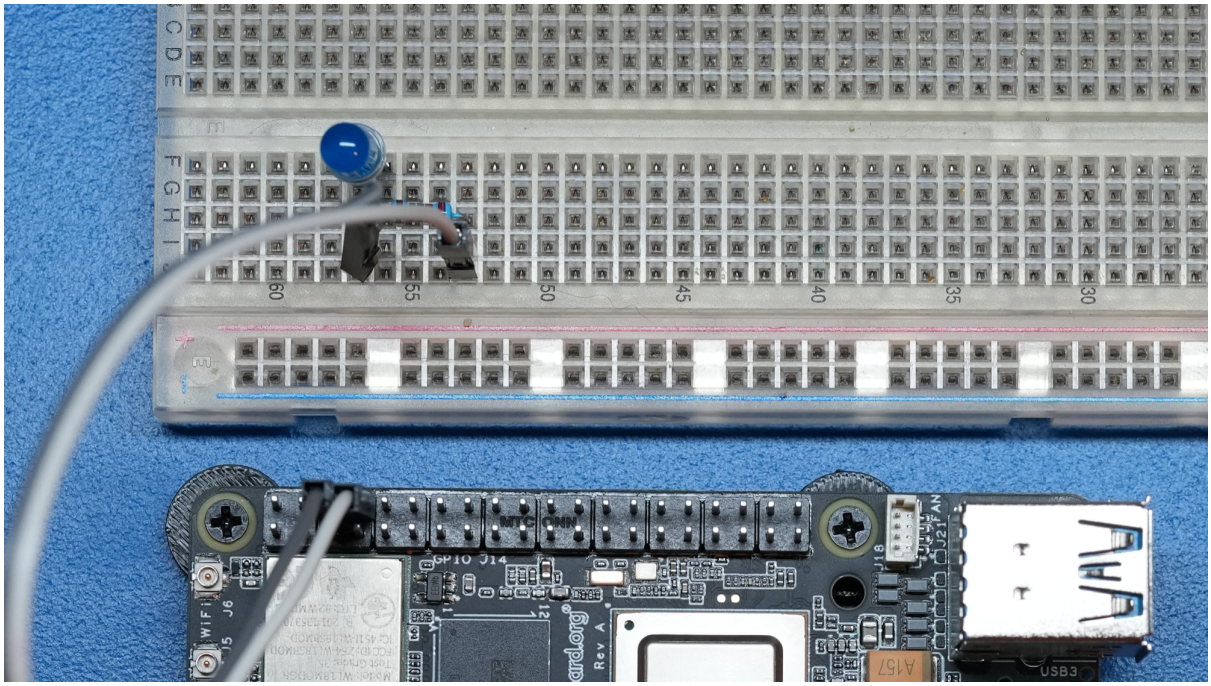


Fig. 5.4: GPIO OFF state

(continued from previous page)

```
    gpioset $(gpiofind GPIO14)=0
    sleep 1
done
```

- Close the editor by pressing `Ctrl + O` followed by `Enter` to save the file and then press to `Ctrl + X` exit
- Now execute the `blinky.sh` script by typing:

```
bash blinky.sh
```

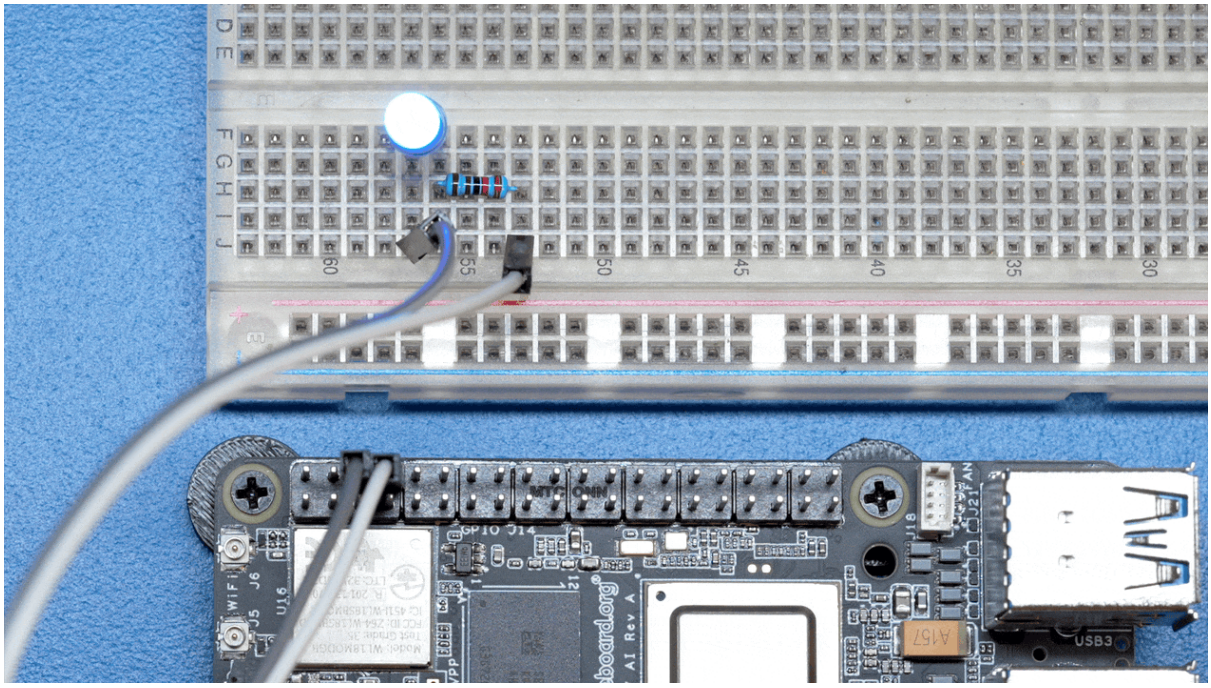


Fig. 5.5: LED blinking

- You can exit the `blinky.sh` program by pressing `CTRL + C` on your keyboard.

Understanding the code

```
#!/bin/bash

while :
do
   gpioset $(gpiofind GPIO14)=1 ①
   sleep 1 ②
   gpioset $(gpiofind GPIO14)=0 ③
   sleep 1 ④
done
```

The script is an infinite `while` loop in which we do the following:

① set the HAT Pin 8 (GPIO14) as 1 (HIGH)

② Wait 1 Second

③ set the HAT Pin 8 (GPIO14) as 0 (LOW)

④ Wait 1 Second

5.1.5 Blink an LED using Python

Using python you can blink an LED.

The `libgpiod` should already be installed in the default image, in case it's not installed then install it by using the command below.

```
sudo apt-get install python3-libgpiod
```

Below is the Python code to blink an LED connected to GPIO14.

Listing 5.1: blinky.py

```
import gpiod
import time

gpio14 = gpiod.find_line('GPIO14')
gpio14.request(consumer='beagle', type=gpiod.LINE_REQ_DIR_OUT, default_val=0)

while True:
   gpio14.set_value(1)
   time.sleep(1)
   gpio14.set_value(0)
   time.sleep(1)
```

To open the Python REPL, type `python` in the terminal like below. Copy the above code and paste it into the terminal. After running the code, the LED connected to GPIO14 will start blinking.

```
debian@BeagleBone:~$ python
Python 3.11.2 (main, May  2 2024, 11:59:08) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Alternatively, you can create a Python script to achieve the same effect. Create a new file named `blinky.py` and open it using the `nano` editor with the following command, copy and paste the above code to the file.

```
nano blinky.py
```

Press `CTRL+O` & `ENTER` to save the `blinky.py` script and then `CTRL+X` to exit.

To run the `blinky.py` script execute the command below,

```
python blinky.py
```

After running the code you can see the LED connected to `GPIO14` or `HAT Pin 08` blinking.

Understanding the code

```
import gpiod ①
import time ②

gpio14 = gpiod.find_line('GPIO14') ③
gpio14.request(consumer='beagle', type=gpiod.LINE_REQ_DIR_OUT, default_
↪val=0) ④

while True: ⑤
   gpio14.set_value(1) ⑥
   time.sleep(1) ⑦
   gpio14.set_value(0) ⑧
   time.sleep(1) ⑨
```

① Importing `gpiod` module

② Importing `time` module

③ Linking GPIO14 pin object to gpio14 variable for better accessibility

④ Set GPIO14 as OUTPUT pin

⑤ Create infinite `while` loop

⑥ Set the GPIO14 as 1 (HIGH)

⑦ Wait 1 Second

⑧ Set the GPIO14 as 0 (LOW)

⑨ Wait 1 Second

5.1.6 Read a Button

A push button simply completes an electric circuit when pressed. Depending on wiring, it can drive a signal either "Low" (GND) or "High" (3.3V).

We will connect our Button between HAT Pin 12 (GPIO18) and Ground (GND).

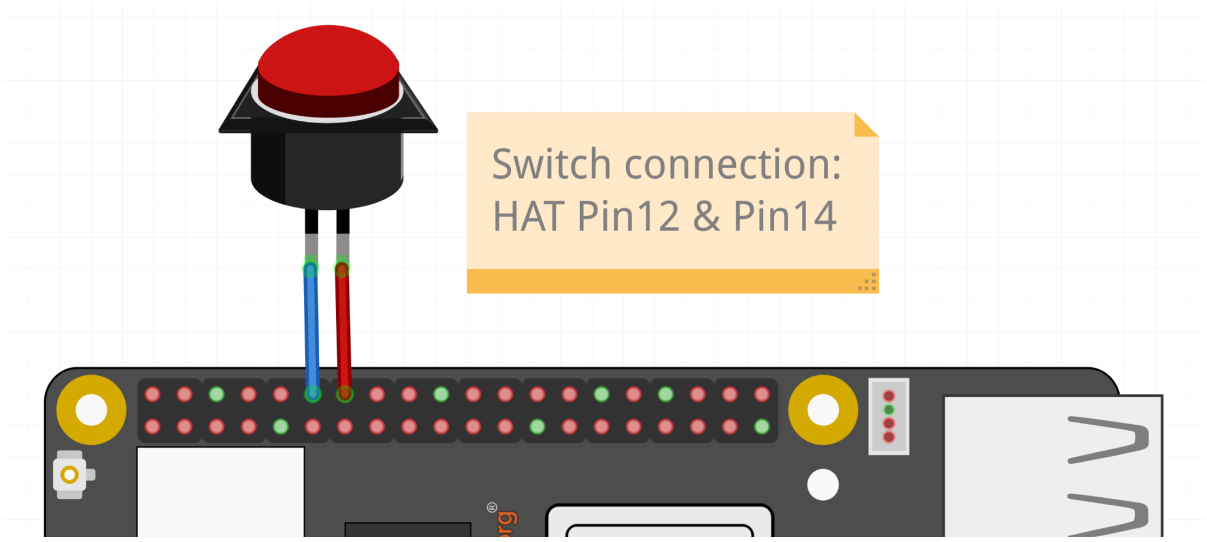


Fig. 5.6: Button connected to HAT Pin12 (GPIO18)

The cool part is since we have an internal pull-up resistor, we don't need an external one! The pull resistor guarantees that the Pin stays in a known (HIGH) state unless the button is pressed, in which case it will go LOW.

- Reading GPIOs can be done using the `gpioget` command

```
gpioget --bias=pull-up $(gpiofind GPIO18)
```

Results in `1` if the Input is held `HIGH` or `0` if the Input is held `LOW`

Let's create a script called `button.sh` to continuously read an input pin connected to a button and print out when it's pressed!

- Create the file,

```
touch button.sh
```

- Open the file using `nano` editor,

```
nano button.sh
```

- Copy paste the code below to `button.sh` file,

```
#!/bin/bash

while :
do
   if (( $(gpioget --bias=pull-up $(gpiofind GPIO18)) == 0))
   then
      echo "Button Pressed!"
   fi
done
```

- Close the editor by pressing `Ctrl + O` followed by `Enter` to save the file and then press to `Ctrl + X` exit
- Now execute the `button.sh` script by typing:

```
bash button.sh
```

- You can exit the `button.sh` by pressing `Ctrl + C` on your keyboard.

5.1.7 Combining the Two

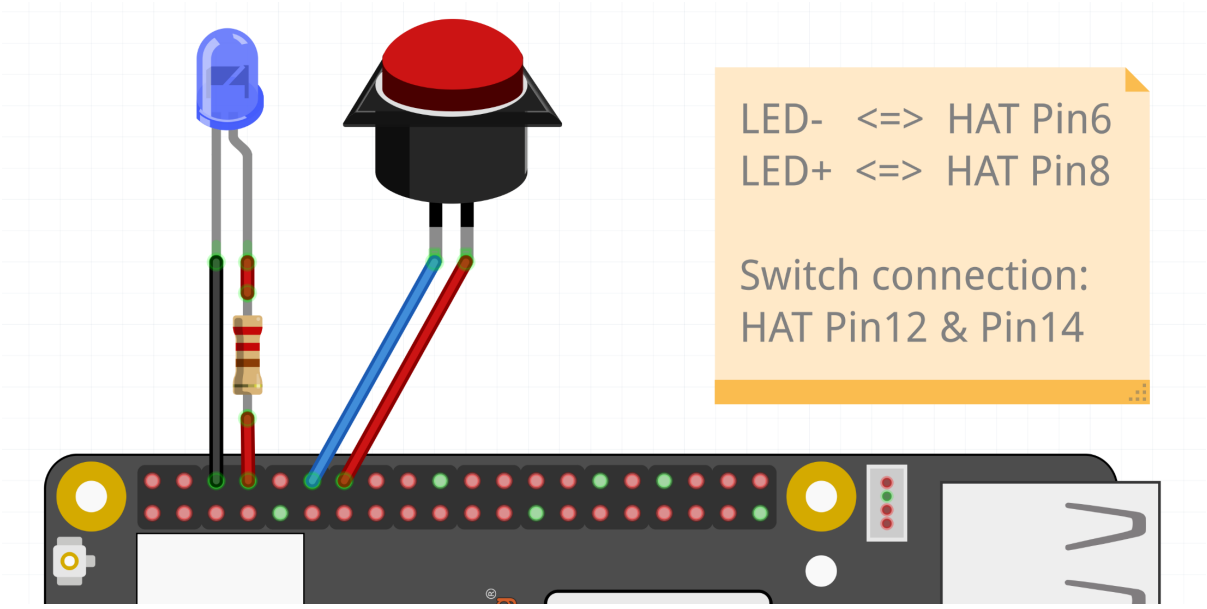


Fig. 5.7: Button connected to HAT Pin12 (GPIO18) & LED connected to HAT Pin8 (GPIO14)

Now, logically, let's make an LED match the state of the button.

Let's create a script called **blinkyButton.sh**:

- Create the file,

```
touch blinkyButton.sh
```

- Open the file using `nano` editor,

```
nano blinkyButton.sh
```

- Copy paste the code below to `blinkyButton.sh` file,

```
#!/bin/bash

while :
do
   if (( $(gpioget --bias=pull-up $(gpiofind GPIO18)) == 0))
   then
      gpioset $(gpiofind GPIO14)=1
   else
      gpioset $(gpiofind GPIO14)=0
   fi
done
```

- Close the editor by pressing `Ctrl + O` followed by `Enter` to save the file and then press to `Ctrl + X` exit
- Now execute the `blinkyButton.sh` script by typing:

```
bash blinkyButton.sh
```

This means when we see HAT Pin 12 (GPIO18) go LOW, we know the button is pressed, so we set HAT Pin 8 (GPIO14) (our LED) to ON, otherwise, we turn it OFF.

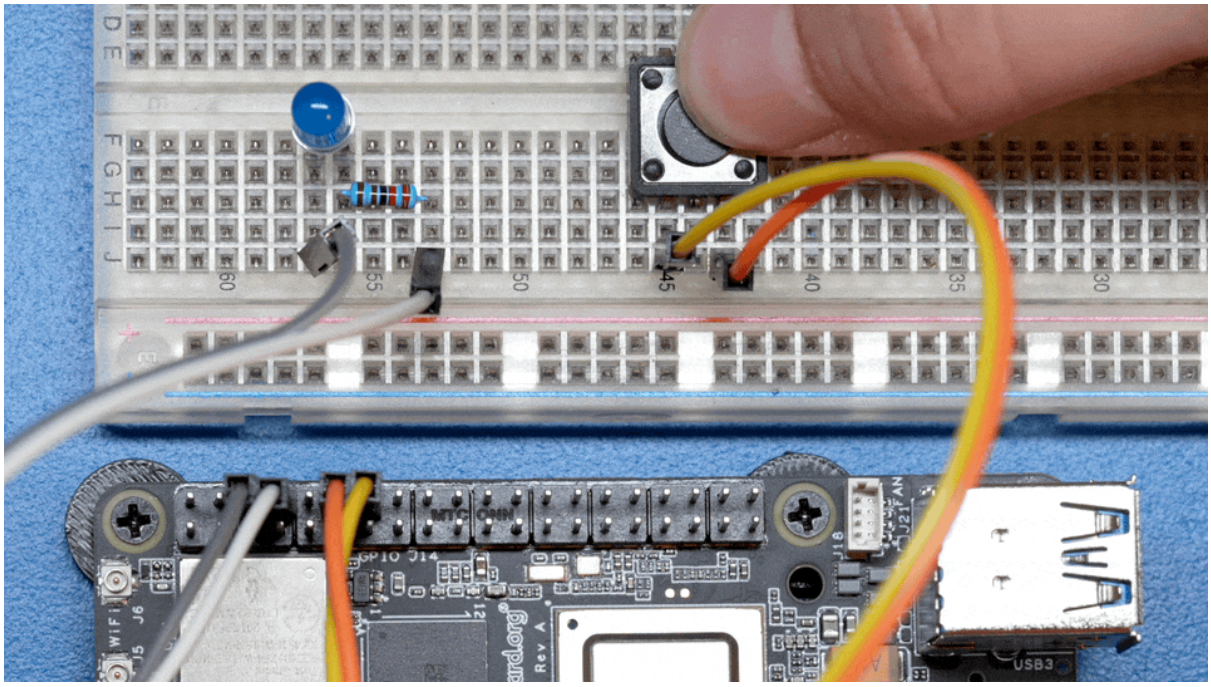


Fig. 5.8: LED is ON when button is pressed

- You can exit the `blinkyButton.sh` program by pressing `Ctrl + C` on your keyboard.

5.1.8 Understanding Internal Pull Resistors

Pull-up and pull-down resistors are used in digital circuits to ensure that inputs to logic settle at expected levels.

- `Internal pull-up resistors` connects the pin to a high voltage level (e.g., 3.3V) to ensure the pin input reads as a logic high (1) when no active device is pulling it low.
- `Internal pull-down resistors` connects the pin to ground (GND) to ensure the input reads as a logic low (0) when no active device is pulling it high.

These resistors prevent floating inputs and undefined states.

By default, all GPIOs on the HAT Header are configured as **Inputs with Pull-up Resistors Enabled**.

This is important for something like a button, as without it, once a button is released, it goes in an "undefined" state!

To configure Pull-ups on a per-pin basis, we can use pass the following arguments within **gpioget or gpioset**:

```
-B, --bias=[as-is|disable|pull-down|pull-up] (defaults to 'as-is')
```

The "Bias" argument has the following options:

- **as-is** - This leaves the bias as-is... quite self explanatory
- **disable** - This state is also known as High-Z (high impedance) where the Pin is left Floating without any bias resistor
- **pull-down** - In this state, the pin is pulled DOWN by the internal 50KΩ resistor
- **pull-up** - In this state, the pin is pulled UP by the internal 50KΩ resistor

For example, a command to read an input with the Bias intentionally disabled would look like this:

```
gpioget --bias=disable $(gpiofind GPIO14)
```

Pull resistors are a foundational block of digital circuits and understanding when to (and not to) use them is important.

This article from SparkFun Electronics is a good basic primer - Link

5.1.9 Troubleshooting

- **My script won't run!**

Make sure you gave the script execute permissions first and that you're executing it with a `./` before

- To make it executable:

```
chmod +X scriptName.sh
```

- To run it:

```
./scriptName.sh
```

5.1.10 Bonus - Turn all GPIOs ON/OFF

- Copy and paste this with the button on the right to turn **all pins ON**.

```
gpioset $(gpiofind GPIO14)=1 ;\ gpioset $(gpiofind GPIO15)=1 ;\ gpioset
↪$(gpiofind GPIO17)=1 ;\ gpioset $(gpiofind GPIO18)=1 ;\ gpioset $(gpiofind␣
↪GPIO27)=1 ;\ gpioset $(gpiofind GPIO22)=1 ;\ gpioset $(gpiofind GPIO23)=1 ;\
↪ gpioset $(gpiofind GPIO24)=1 ;\ gpioset $(gpiofind GPIO10)=1 ;\ gpioset
↪$(gpiofind GPIO9)=1 ;\ gpioset $(gpiofind GPIO25)=1 ;\ gpioset $(gpiofind␣
↪GPIO11)=1 ;\ gpioset $(gpiofind GPIO8)=1 ;\ gpioset $(gpiofind GPIO7)=1 ;\␣
↪gpioset $(gpiofind GPIO1)=1 ;\ gpioset $(gpiofind GPIO6)=1 ;\ gpioset
↪$(gpiofind GPIO12)=1 ;\ gpioset $(gpiofind GPIO13)=1 ;\ gpioset $(gpiofind␣
↪GPIO19)=1 ;\ gpioset $(gpiofind GPIO16)=1 ;\ gpioset $(gpiofind GPIO26)=1 ;\
↪ gpioset $(gpiofind GPIO21)=1
```

- Similarly, copy and paste this to turn **all pins OFF**.

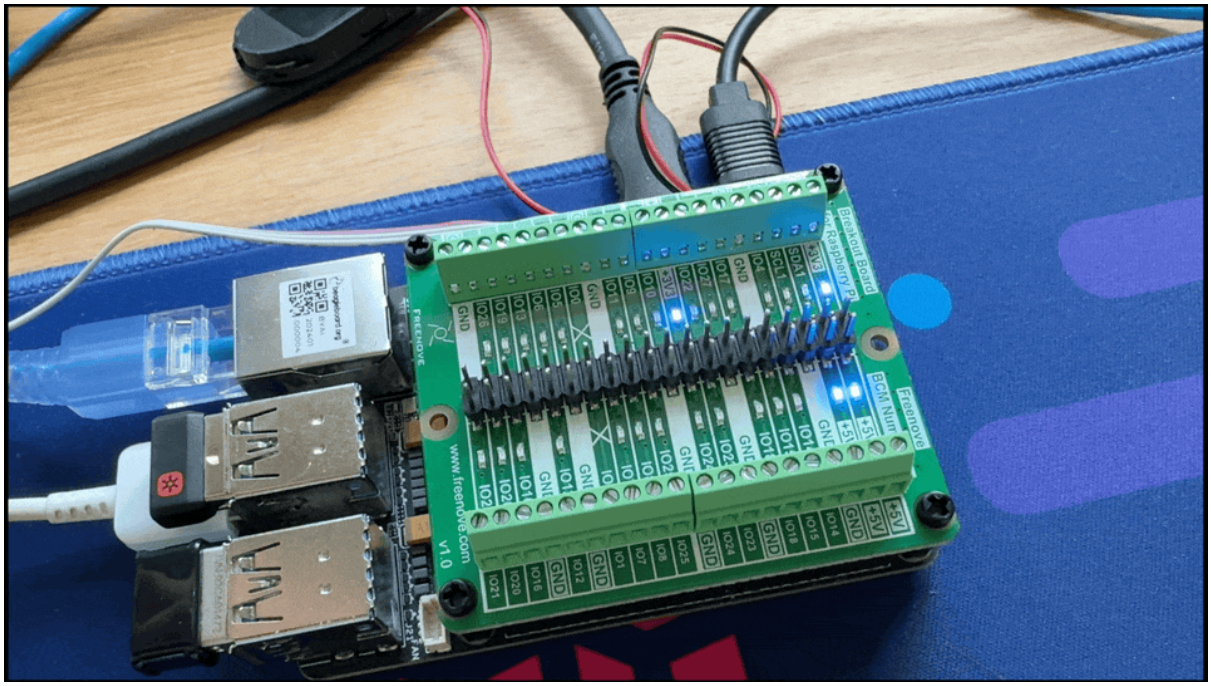


Fig. 5.9: All HAT GPIO toggle

```
gpioset $(gpiofind GPIO14)=0 ;\ gpioset $(gpiofind GPIO15)=0 ;\ gpioset
↪$(gpiofind GPIO17)=0 ;\ gpioset $(gpiofind GPIO18)=0 ;\ gpioset $(gpiofind␣
↪GPIO27)=0 ;\ gpioset $(gpiofind GPIO22)=0 ;\ gpioset $(gpiofind GPIO23)=0 ;\
↪ gpioset $(gpiofind GPIO24)=0 ;\ gpioset $(gpiofind GPIO10)=0 ;\ gpioset
↪$(gpiofind GPIO9)=0 ;\ gpioset $(gpiofind GPIO25)=0 ;\ gpioset $(gpiofind␣
↪GPIO11)=0 ;\ gpioset $(gpiofind GPIO8)=0 ;\ gpioset $(gpiofind GPIO7)=0 ;\␣
↪gpioset $(gpiofind GPIO1)=0 ;\ gpioset $(gpiofind GPIO6)=0 ;\ gpioset
↪$(gpiofind GPIO12)=0 ;\ gpioset $(gpiofind GPIO13)=0 ;\ gpioset $(gpiofind␣
↪GPIO19)=0 ;\ gpioset $(gpiofind GPIO16)=0 ;\ gpioset $(gpiofind GPIO26)=0 ;\
↪ gpioset $(gpiofind GPIO21)=0
```

5.1.11 Going Further

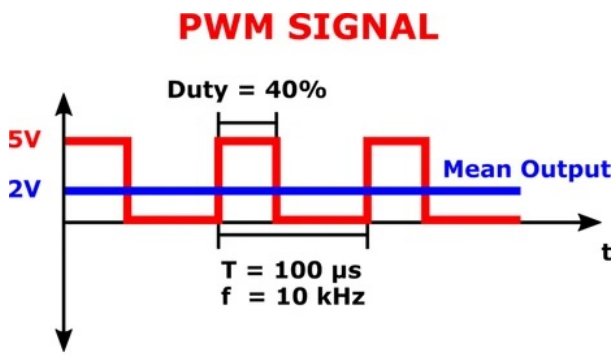
- pinout.beagley.ai
- GPIOSet Documentation
- GPIOGet Documentation

5.2 Pulse Width Modulation (PWM)

Todo: Add further testing steps, results, and images..

5.2.1 What is it

`PWM` or `Pulse Width Modulation`, is a technique used to control the amount of power delivered to an electronic device by breaking up the power signal into discrete ON and OFF periods. The amount of time the signal spends ON during each cycle determines the output power level (brightness of the LED).



5.2.2 Configuring PWM overlay

To enable any of the PWM Pins, we have to modify the following file: `/boot/firmware/extlinux/extlinux.conf`. We can check the available list of Device Tree Overlays using command below,

```
ls /boot/firmware/overlays/ | grep "beagley-ai-pwm"
```

When executed the above command should give output show below,

```
debian@BeagleBone:~$ ls /boot/firmware/overlays/ | grep "beagley-ai-pwm"
k3-am67a-beagley-ai-pwm-ecap0-gpio12.dtbo
k3-am67a-beagley-ai-pwm-ecap1-gpio16.dtbo
k3-am67a-beagley-ai-pwm-ecap1-gpio21.dtbo
k3-am67a-beagley-ai-pwm-ecap2-gpio17.dtbo
k3-am67a-beagley-ai-pwm-ecap2-gpio18.dtbo
k3-am67a-beagley-ai-pwm-epwm0-gpio12.dtbo
k3-am67a-beagley-ai-pwm-epwm0-gpio14.dtbo
k3-am67a-beagley-ai-pwm-epwm0-gpio15.dtbo
k3-am67a-beagley-ai-pwm-epwm0-gpio5.dtbo
k3-am67a-beagley-ai-pwm-epwm1-gpio13.dtbo
k3-am67a-beagley-ai-pwm-epwm1-gpio20.dtbo
k3-am67a-beagley-ai-pwm-epwm1-gpio21.dtbo
k3-am67a-beagley-ai-pwm-epwm1-gpio6.dtbo
```

Using hat-08 (GPIO14) as pwm

Add the line shown below to `/boot/firmware/extlinux/extlinux.conf` file to load the gpio14 pwm device tree overlay:

```
fdtoverlays /overlays/k3-am67a-beagley-ai-pwm-epwm0-gpio14.dtbo
```

Your `/boot/firmware/extlinux/extlinux.conf` file should look something like this:

```
label microSD (default)
   kernel /Image
   append console=ttyS2,115200n8 root=/dev/mmcblk1p3 ro rootfstype=ext4␣
↪resume=/dev/mmcblk1p2 rootwait net.ifnames=0 quiet
   fdtdir /
   fdt /ti/k3-am67a-beagley-ai.dtb
   fdtoverlays /overlays/k3-am67a-beagley-ai-pwm-epwm0-gpio14.dtbo
   initrd /initrd.img
```

Now reboot you BeagleY-AI to load the overlay,

```
sudo reboot
```

5.2.3 How do we do it

To configure HAT pin8 (GPIO14) PWM symlink pin using `beagle-pwm-export` execute the command below,

```
sudo beagle-pwm-export --pin hat-08
```

Let's create a script called `fade.sh` that cycles through LED brightness on HAT pin8 by changing PWM duty cycle.

```
touch fade.sh
```

Now open the file with nano editor,

```
nano fade.sh
```

In the editor copy paste the script content below,

```
#!/bin/bash

PWMPIN="/dev/hat/pwm/GPIO14"


echo 1000 > $PWMPIN/period
echo 0 > $PWMPIN/duty_cycle
echo 0 > $PWMPIN/enable
sleep 1

for i in {1..500};
do
        echo $i > $PWMPIN/duty_cycle
        echo 1 > $PWMPIN/enable
        echo $i
        sleep 0.0005
done

for i in {500..1};
do
    echo $i > $PWMPIN/duty_cycle
    echo 1 > $PWMPIN/enable
    echo $i
    sleep 0.0005
done
```

- Close the editor by pressing `Ctrl + O` followed by `Enter` to save the file and then press to `Ctrl + X` exit
- Now execute the `fade.sh` script by typing:

```
bash fade.sh
```

- You can exit the `fade.sh` program by pressing `Ctrl + C` on your keyboard.

Todo: Add section about driving Servo Motors at 50KHz

5.2.4 Troubleshooting

Todo: Fill out empty section

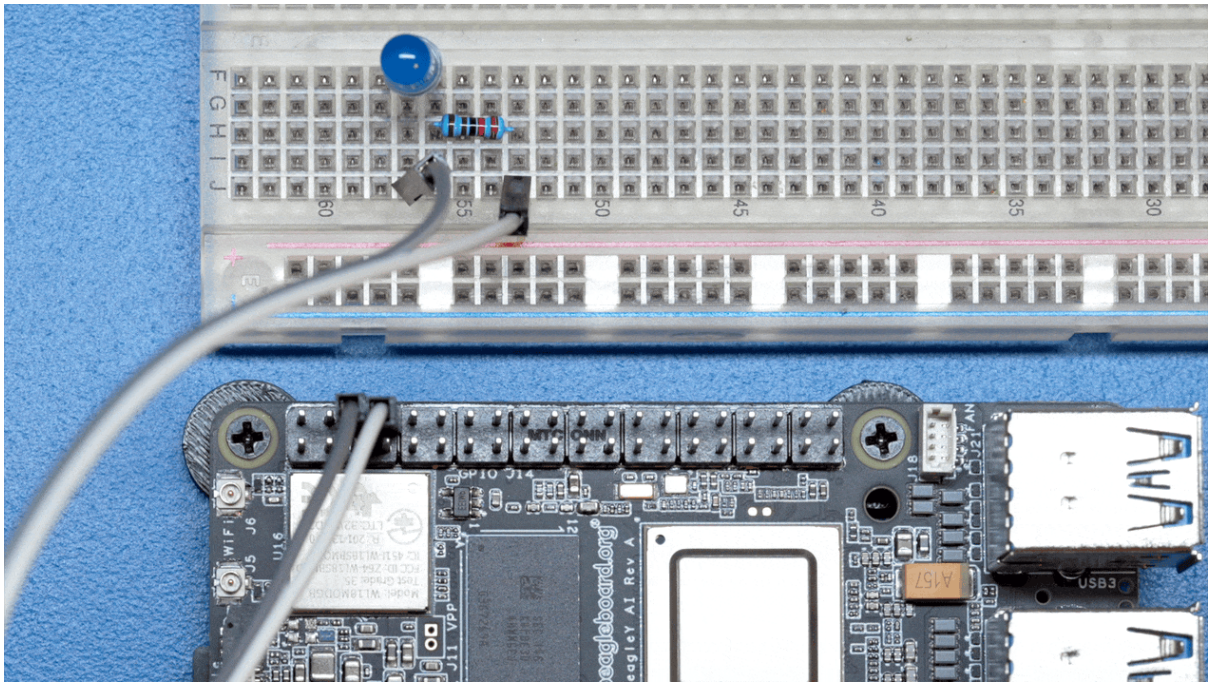


Fig. 5.10: LED PWM fade demo

5.2.5 Going Further

Todo: Fill out empty section

5.3 Using the on-board Real Time Clock (RTC)

Real Time Clocks (RTCs) provide precise and reliable timekeeping capabilities, which are beneficial for applications ranging from simple timekeeping to complex scheduling and secure operations.

Without an RTC, a computer must rely on something called Network Time Protocol (NTP) to obtain the current time from a network source. There are many cases however where an SBC such as BeagleY-AI may not have a constant or reliable network connection. In situations such as these, an RTC allows the board to keep time even if the network connection is severed or the board loses power for an extended period of time.

Fortunately, BeagleY-AI comes with a built-in DS1340 RTC for all your fancy time keeping needs!

5.3.1 Required Hardware

BeagleY provides a **1.00 mm pitch, 2-pin JST SH connector** for a coin cell battery to enable the RTC to keep time even if power is lost to the board.

These batteries are available from several vendors:

- Raspberry Pi 5 RTC Battery via Adafruit
- Raspberry Pi 5 RTC Battery via DigiKey
- CR2023 battery holder for Pi 5 via Amazon

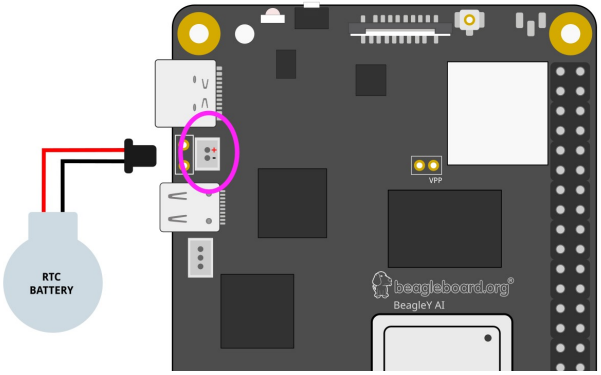


Fig. 5.11: BeagleY-AI RTC battery connection

5.3.2 Uses for an RTC

1. **Maintaining Accurate Time:** RTCs provide an accurate clock that continues to run even when the SBC is powered down. This is crucial for maintaining the correct time and date across reboots.
2. **Timestamping:** Many applications need to know the current time for timestamping data, logs, or events. For example, IoT devices may need to log sensor data with precise timestamps.
3. **Scheduling Tasks:** In some applications, tasks need to be scheduled at specific times. An RTC allows the SBC to keep track of time accurately, ensuring that tasks are performed at the correct times.
4. **Network Synchronization:** If the SBC is part of a larger network, having an accurate time helps with synchronizing data and events across the network.
5. **Standby Power Efficiency:** Many RTCs operate with a very low power requirement and can keep time even when the rest of the board is in a low-power or sleep mode. This helps in reducing overall power consumption.

5.3.3 Setting time

Note: You must set the time before being able to read it. If you don't do this first, you'll see errors. You may connect your BeagleY-AI to a network so it can get time from an NTP server.

You can set time manually by running the following command:

```
sudo hwclock --set --date "2024-06-11 22:22:22"
```

5.3.4 Diving Deeper

There are actually two different "times" that your Linux system keeps track of.

- System time, which can be read using the `date` or `timedatectl` commands
- RTC (hardware) time which can be read using the `hwclock` command shown above.

Open up a BeagleY-AI console and try the commands shown below,

- Reading the current **system time** is achieved using the `date` command,

```
date
```

The `date` command should print `Tue Jun 11 06:30:51 UTC 2024`.

- Reading the current **RTC (hardware) time** is achieved using the `hwclock` command.

```
sudo hwclock
```

The `hwclock` command should print `2024-05-10 00:00:02.224187-05:00`.

Comparing both `date` and `hwclock` output above we see the time format is different. we add some extra instructions to match the format.

```
debian@BeagleBone:~$ date +%Y-%m-%d' '%H:%M:%S.%N%:z
2024-05-10 21:06:50.058595373+00:00

debian@BeagleBone:~$ sudo hwclock
2024-05-10 21:06:56.692874+00:00
```

But why?

We see here that our system and hardware clock are over 9 seconds apart!

Ok, in this particular case we set the HW clock slightly ahead to illustrate the point, but in real life "drift" is a real problem that has to be dealt with. Environmental conditions like temperature or stray cosmic rays can cause electronics to become ever so slightly out of sync, and these effects only grow over time unless corrected. It's why RTCs and other fancier time keeping instruments implement various methods to help account for this such as temperature compensated oscillators.

Let's fix our hardware clock. We assume here that the system clock is freshly synced over NTP so it's going to be our true time "source".

```
sudo hwclock --systohc
```

Let's create a simple script to get the two times, we'll call it `getTime.sh`,

```
nano getTime.sh
```

copy and paste the below code in that file,

```
HWTIME=$(sudo hwclock)
echo "RTC - ${HWTIME} "

SYSTIME=$(date +%Y-%m-%d' '%H:%M:%S.%N%:z)
echo "SYS - ${SYSTIME} "
```

Now let's run it!

```
bash getTime.sh
```

The script gives us this output,

> RTC - 2024-05-10 21:52:58.374954+00:00
>
> SYS - 2024-05-10 21:52:59.048442940+00:00

As we can see, we're still about a second off, but this is because it takes a bit of time to query the RTC via I2C.

If you want to learn more, the *Going Further* at the end of this article is a good starting point!

5.3.5 Troubleshooting

The most common error results from not having initialized the RTC at all. This usually happens if the system is powered on without an RTC battery and without a network connection.

In such cases, you should be able to read the time after setting the time as follows:

- Sync clock

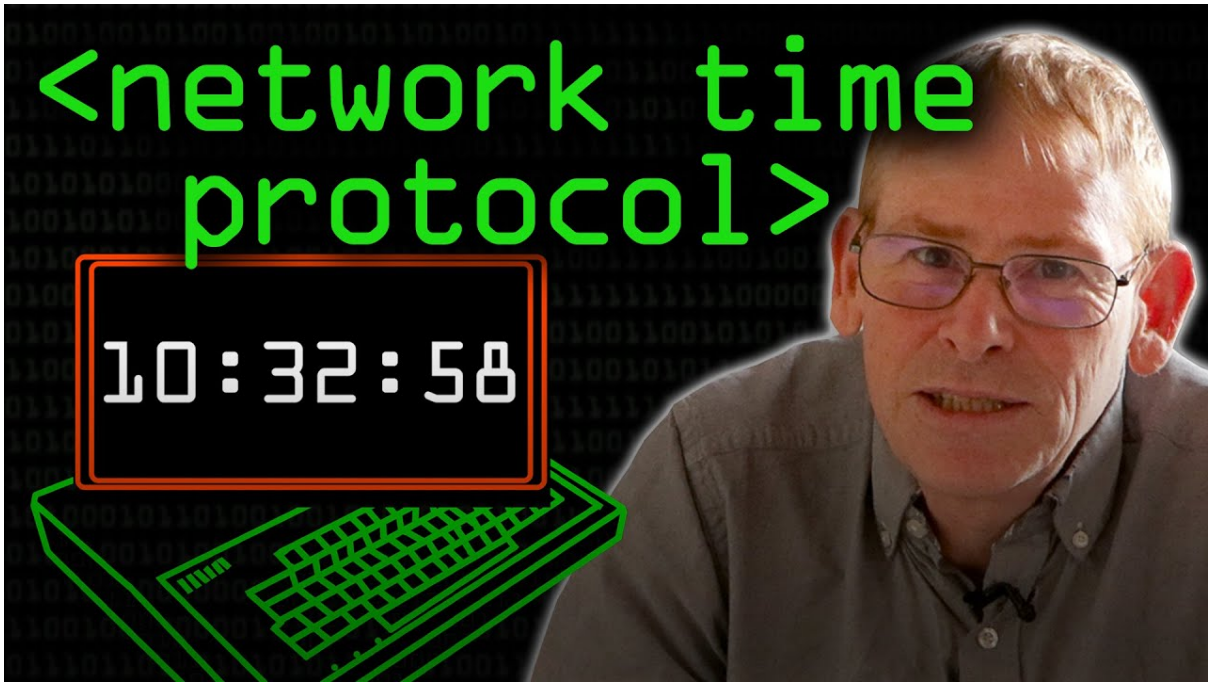


Fig. 5.12: https://youtu.be/BAo5C2qbLq8

```
sudo hwclock --systohc
```

- Check RTC time

```
sudo hwclock
```

The above command should output `2024-05-10 21:06:56.692874+00:00`.

5.3.6 Going Further

Consider learning about topics such as time keeping over GPS and Atomic Clocks!

There are some good YouTube videos below to provide sources for inspiration.

Network Time Protocol - Computerphile

Nanosecond Clock Sync - Jeff Geerling

Using GPS with PPS to synchronize clocks over the network

5.4 Using I2C OLED Display

Using I2C on BeagleY-AI is similar to using it on any Raspberry Pi compatible board. The image below shows the BeagleY-AI I2C pinout. For more information check pinout.beagley.ai/pinout/i2c.

5.4.1 OLED (ssd1306) displays

Fig. 5.13: https://youtu.be/RvnG-ywF6_s

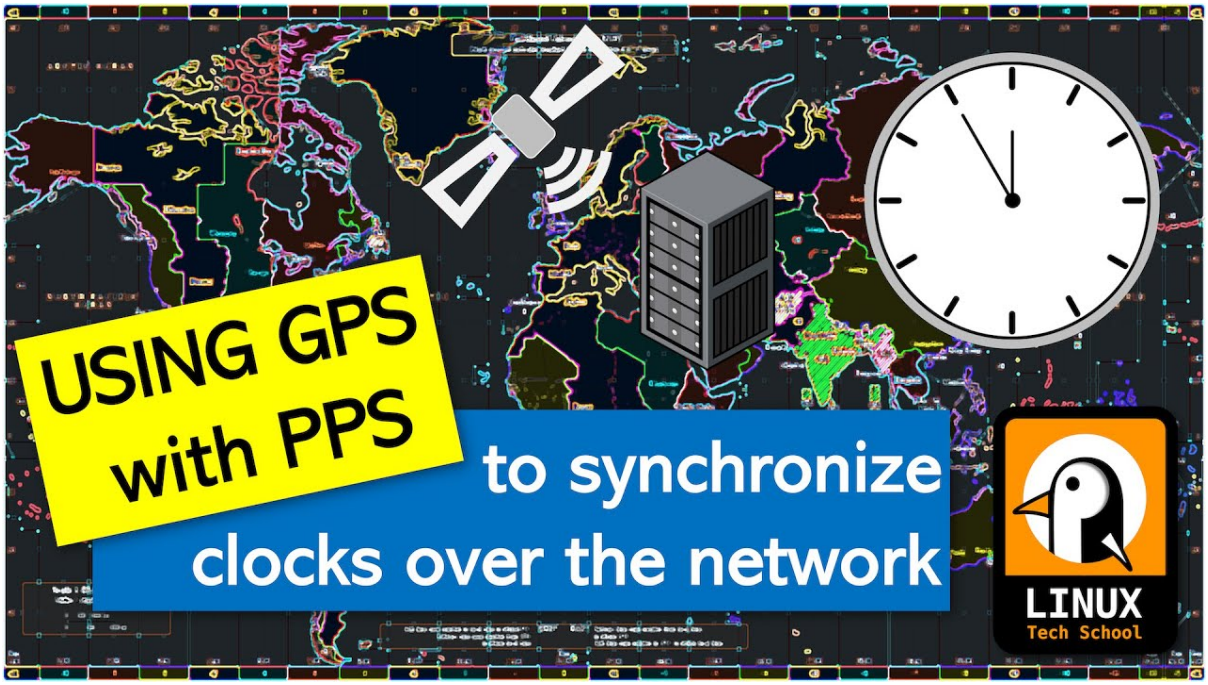


Fig. 5.14: https://youtu.be/7aTZ66ZL6Dk

Fig. 5.15: BeagleY-AI I2C pinout

Wiring/connection

Following the I2C pinout shown above let's make the connection of our OLED display with BeagleY-AI. Connection for both `128x64` and `128x32` resolution displays are demonstrated in the images below:

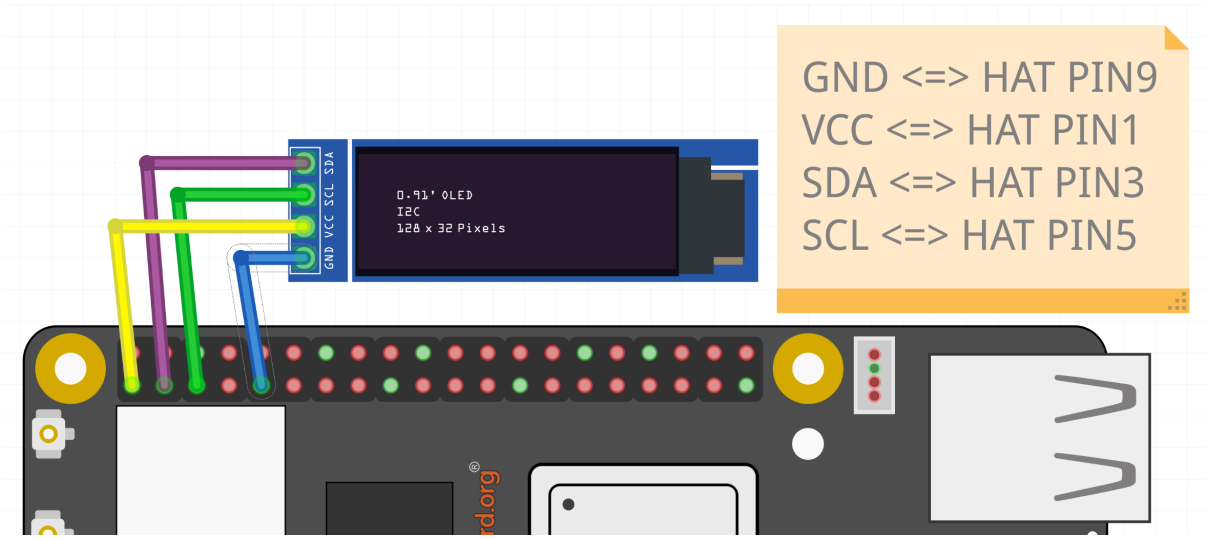


Fig. 5.16: OLED display 128x32

To check if your OLED is correctly connected to your BeagleY-AI you can use `i2cdetect` command as shown below.

```
i2cdetect -y -r 1
```

The above command should show `3c` address occupied in the output, which is the default I2C address of our OLED display.

```
debian@BeagleBone:~$ i2cdetect -y -r 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                         -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- 3c -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

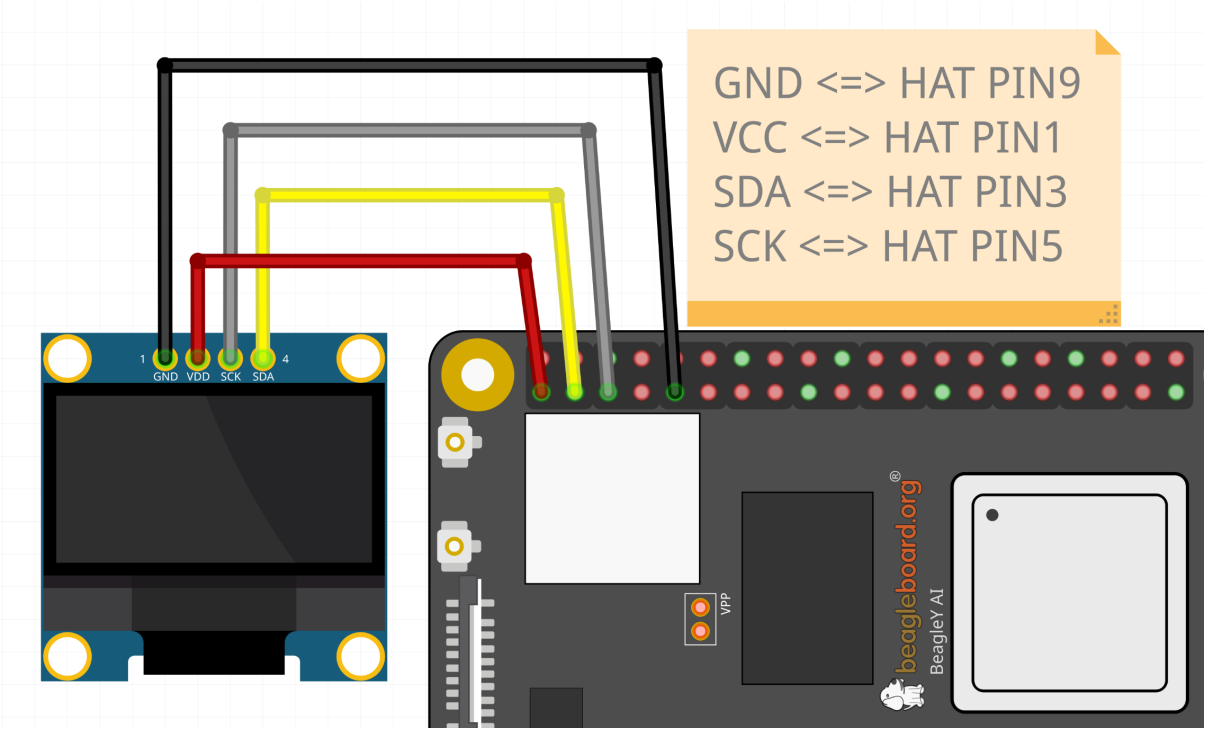


Fig. 5.17: OLED display 128x64

Using kernel driver

To use the kernel driver to drive the SSD1306 oled, we have created an overlay `/boot/firmware/overlays/k3-am67a-beagley-ai-i2c1-ssd1306.dtbo`. To load the overlay you have to add `fdtoverlays /overlays/k3-am67a-beagley-ai-i2c1-ssd1306.dtbo` to `/boot/firmware/extlinux/extlinux.conf` as shown below.

Note: Current overlay is created for 128x64 OLED displays, you can update the overlay to use it for other resolution OLED displays.

```
...
...
...

label microSD (default)
kernel /Image
append console=ttyS2,115200n8 root=/dev/mmcblk1p3 ro rootfstype=ext4 resume=/
↪dev/mmcblk1p2 rootwait net.ifnames=0 quiet
fdtdir /
fdt /ti/k3-am67a-beagley-ai.dtb
fdtoverlays /overlays/k3-am67a-beagley-ai-i2c1-ssd1306.dtbo
```

After rebooting the board you should see `/dev/fb0` available.

```
debian@BeagleBone:~$ ls /dev | grep fb
fb0
```

To show random noise on the display you can use `/dev/urandom` and feed it to `/dev/fb0`,

```
cat /dev/urandom > /dev/fb0
```

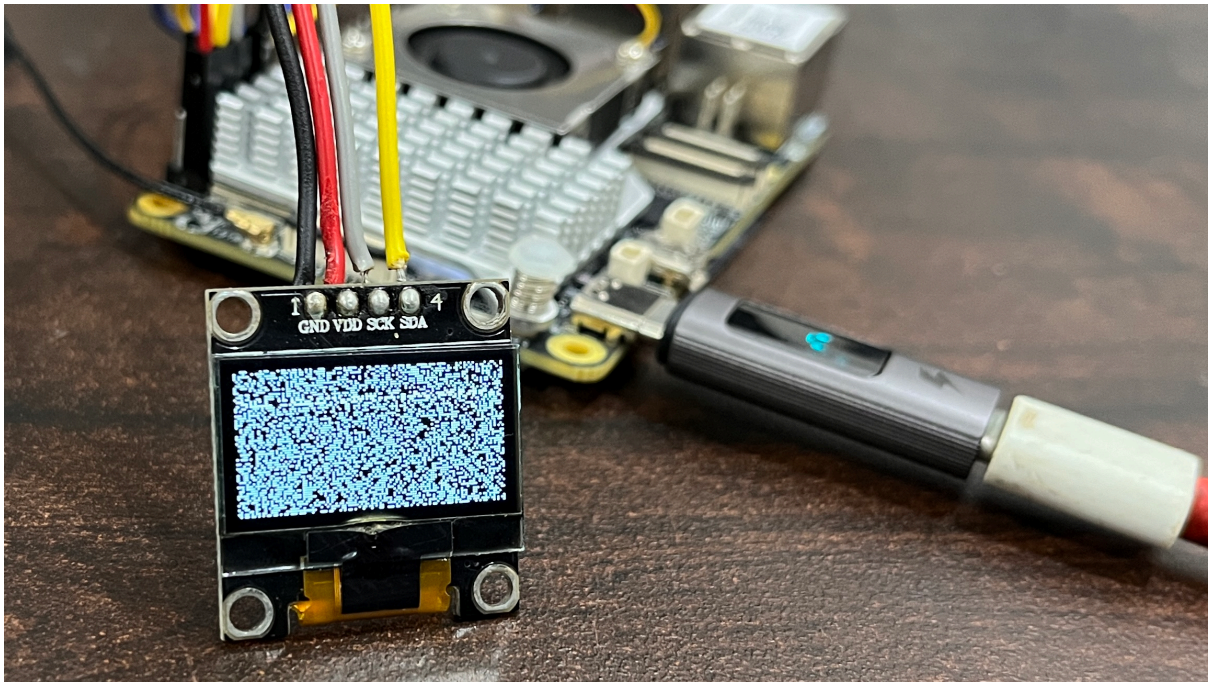


Fig. 5.18: Random noise on SSD1306 OLED

To show blank screen you can use `/dev/zero` and feed it to `/dev/fb0`,

```
cat /dev/zero > /dev/fb0
```

To fill the screen with white pixels you can create a python script called `fill-oled.py` to create `data.out` file and feed it to `/dev/fb0`,

```
nano fill-oled.py
```

Copy paste the below code to `fill-oled.py`,

```
xsize = 128
ysize = 64

with open('data.out', 'wb') as f:
for y in range(0, ysize):
    for x in range(0, xsize):
    pixel = 255
    f.write((pixel).to_bytes(1, byteorder='little'))
```

To get the `data.out` from `fill-oled.py` file execute the command below,

```
python fill-oled.py
```

The above command should create a file called `data.out`. To feed `data.out` to `/dev/fb0` execute the command below,

```
cat data.out > /dev/fb0
```

Todo: Add instructions to use OLED for console and printing text via `/dev/fb0` interface.

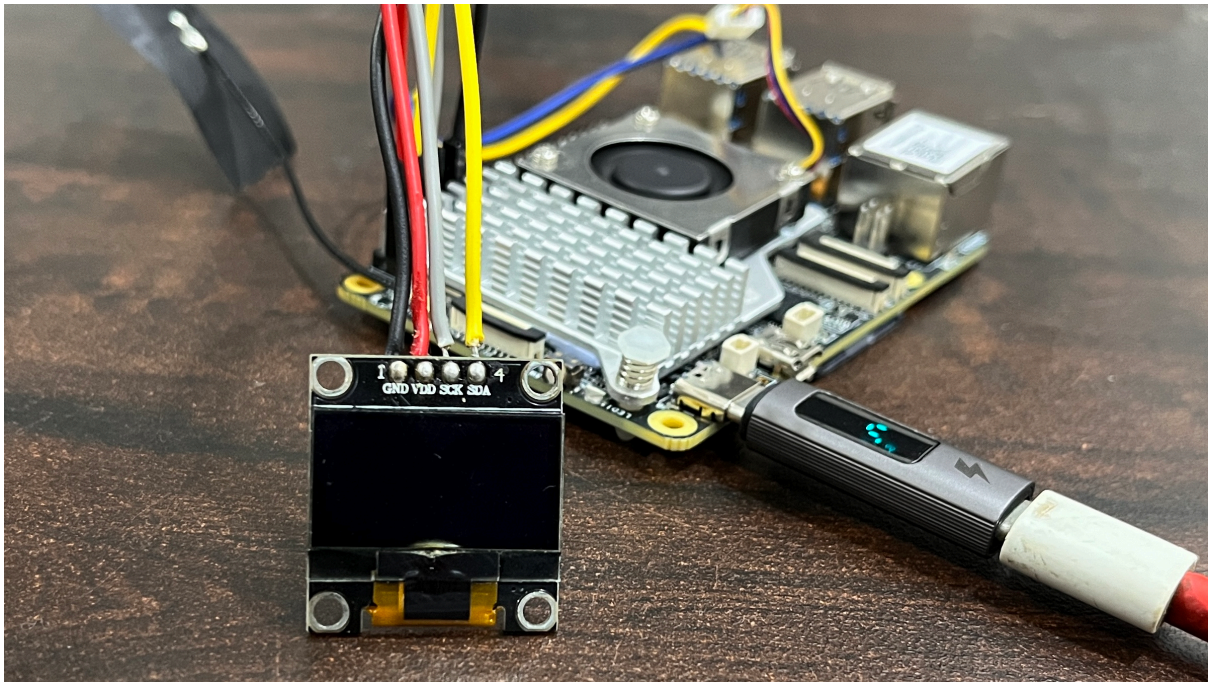


Fig. 5.19: Blank (black/zero) SSD1306 OLED pixels

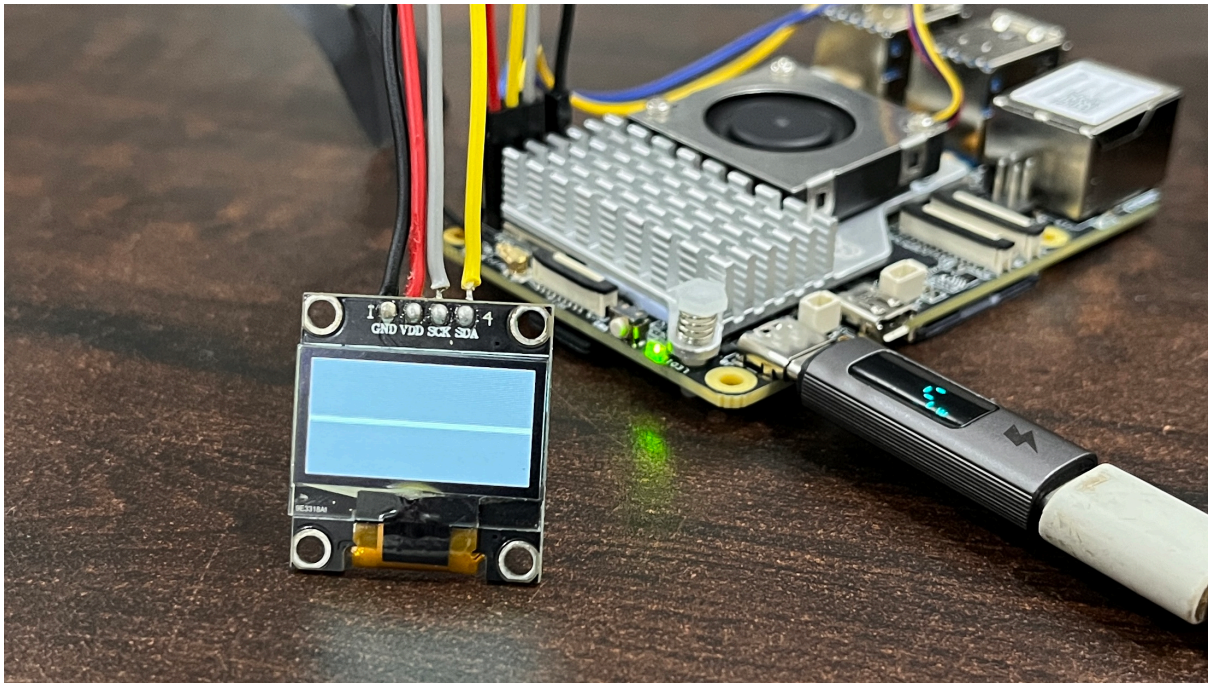


Fig. 5.20: Fill (white/ones) SSD1306 OLED pixels

Setup ssd1306 linux software

There are several examples available online to use OLED (ssd1306) displays under linux. We are using ssd1306_linux from armlabs to demonstrate how you can write to an OLED (ssd1306) display.

Tip: For detailed usage examples of the library check Examples section of the Readme.

Note: If you tried *Using kernel driver*, you must remove the `fdtoverlays /overlays/k3-am67a-beagley-ai-i2c1-ssd1306.dtbo` line from `/boot/firmware/extlinux/extlinux.conf` and `reboot` your BeagleY-AI board before following the instructions provided below.

Clone the `ssd1306_linux` github repository on your BeagleY-AI.

```
git clone https://github.com/armlabs/ssd1306_linux.git
```

Change directory to your cloned `ssd1306_linux` github repository.

```
cd ssd1306_linux
```

Execute `make` to build the binary to control your I2C OLED display.

```
make
```

Now, you should have `ssd1306_bin` binary file generated in the folder that you can use to easily write text on you I2C OLED (ssd1306) display.

Example1: Hello World!!!! let's create a script inside the repository (ssd1306 folder) to print `Hello World!!!!` on the screen.

```
nano hello-world.sh
```

Now copy paste the code shown below in your `hello-world.sh` file. Update the code if your display resolution is not `128x64`, comment out first line and uncomment second line to choose `128x32` display size.

```
./ssd1306_bin -n 1 -I 128x64
#./ssd1306_bin -n 1 -I 128x32


./ssd1306_bin -n 1 -c
./ssd1306_bin -n 1 -r 0
./ssd1306_bin -n 1 -x 1 -y 1
./ssd1306_bin -n 1 -l "Hello World!!!!"
```

Execute the `hello-world.sh` script using command below,

```
source hello-world.sh
```

Executing the command above should print `Hello World!!!!` on your OLED display.

Understanding the code

```
./ssd1306_bin -n 1 -I 128x64 ①
#./ssd1306_bin -n 1 -I 128x32 ②


./ssd1306_bin -n 1 -c ③
```

(continues on next page)

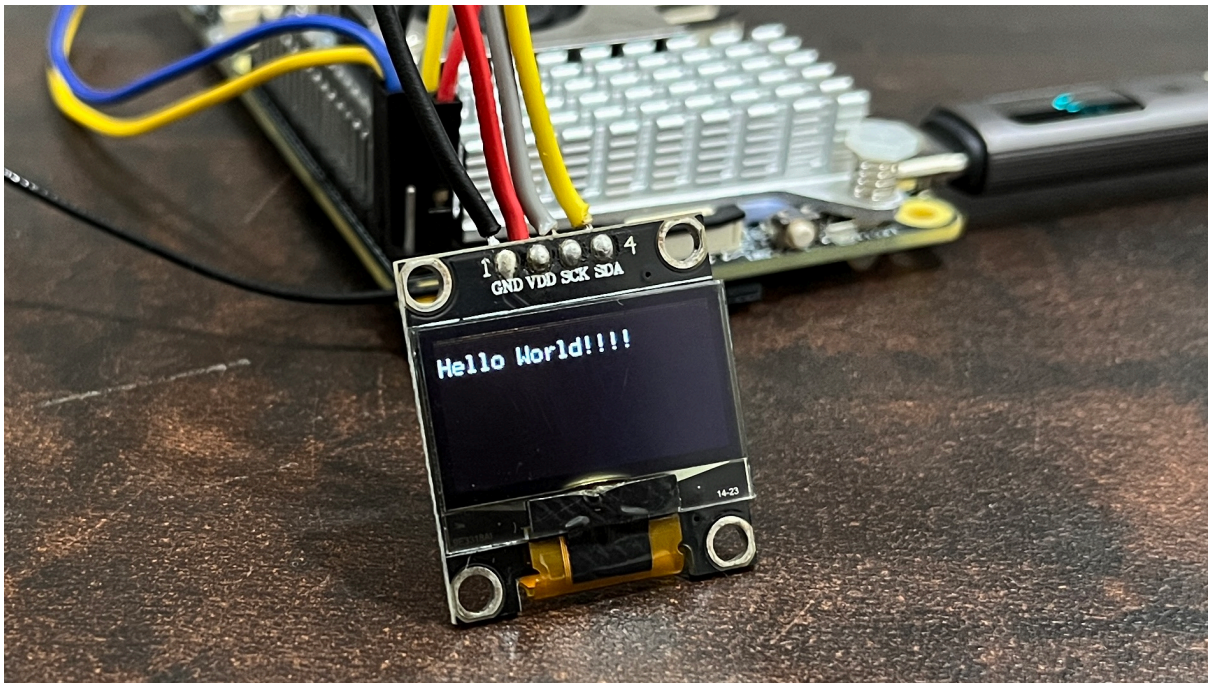


Fig. 5.21: Hello World!!!! on 128x64 OLED

Fig. 5.22: Hello World!!!! on 128x32 OLED

(continued from previous page)

```
./ssd1306_bin -n 1 -r 0 ④
./ssd1306_bin -n 1 -x 1 -y 1 ⑤
./ssd1306_bin -n 1 -l "Hello World!!!!" ⑥
```

① Use this command to set OLED display resolution to `128x64`

② Use this command to set OLED display resolution to `128x32`

③ Clear the display

④ Set rotation to `0/normal`

⑤ Set cursor to location `x:1 y:1`

⑥ Write `Hello World!!!!` to display as line using `-l` command.

Note: We are using `-n 1` because our OLED display is connected to `/dev/i2c-1` port.

Example2: Date and time To print the date and time on our OLED screen we will be using `date` command but you can also use `hwclock` command to show date and time from onboard RTC. For details on using `date` and `hwclock` you can check *Using the on-board Real Time Clock (RTC)* demo.

Let's create `date-time.sh` in the same folder.

```
nano date-time.sh
```

Now copy paste the code shown below in your `date-time.sh` file. Make sure to update the code if your display resolution is not `128x64`, comment out first line and uncomment second line to choose `128x32` display size.

```
./ssd1306_bin -n 1  -I 128x64
#./ssd1306_bin -n 1 -I 128x32

./ssd1306_bin -n 1 -c
./ssd1306_bin -n 1 -r 0

while :
do
./ssd1306_bin -n 1 -x 1 -y 1
./ssd1306_bin -n 1 -f 1 -m "$(date +%Y:%m:%d)\n\n$(date +%H:%M:%S)"
done
```

Execute the `date-time.sh` script using command below,

```
source date-time.sh
```

Executing the command above should print `Date & Time` on your OLED display.

Understanding the code

```
./ssd1306_bin -n 1  -I 128x64 ①
#./ssd1306_bin -n 1 -I 128x32 ②

./ssd1306_bin -n 1 -c ③
./ssd1306_bin -n 1 -r 0 ④

while : ⑤
do
./ssd1306_bin -n 1 -x 1 -y 1 ⑥
./ssd1306_bin -n 1 -f 1 -m "$(date +%Y:%m:%d)\n\n$(date +%H:%M:%S)" ⑦
sleep 0.2 ⑧
done
```

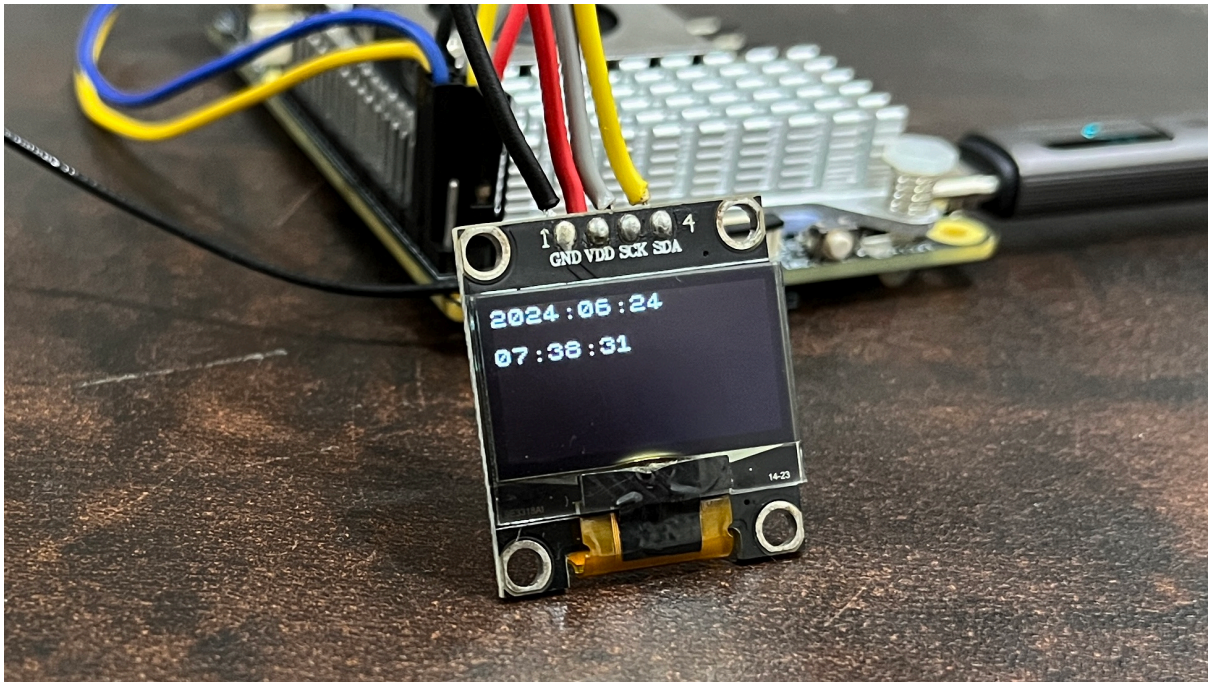


Fig. 5.23: Date & Time on 128x64 OLED

Fig. 5.24: Date & Time on 128x32 OLED

① Use this command to set OLED display resolution to `128x64`

② Use this command to set OLED display resolution to `128x32`

③ Clear the display

④ Set rotation to `0/normal`

⑤ Run infinite loop to regularly update screen.

⑥ Set cursor to location `x:1 y:1`

⑦ Write `Date and Time` to display on separate lines as message using `-m` command.

⑧ Sleep for 200ms (200 milli seconds)

Note: We are using `-n 1` because our OLED display is connected to `/dev/i2c-1` port.

Tip: Other I2C devices can also be connected and used with BeagleY-AI in the same way shown in this demo.

5.5 Using I2C ADC

Using I2C device on BeagleY-AI is similar to using it on any Raspberry Pi compatible board. The image below shows the BeagleY-AI I2C pinout when using 5V for device VCC. For more information check pinout.beagley.ai/pinout/i2c.

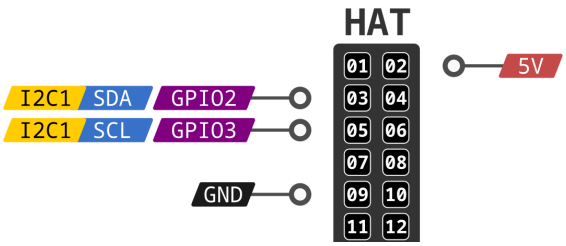


Fig. 5.25: BeagleY-AI I2C pinout (5V)

5.5.1 ADS1115 16-bit ADC

Wiring/connection

Following the I2C pinout shown above let's make the connection of our ADS1115 ADC.

Tip: The ADC can work on `2.0V - 5.5V` voltage range, here we have selected to run it on `5V` from `HAT PIN2`. For more information checkout ADS1115 datasheet.

To check if your ADS1115 ADC is correctly connected to your BeagleY-AI you can use `i2cdetect` command as shown below.

```
i2cdetect -y -r 1
```

The above command should show `48` address occupied in the output, which is the default I2C address of our ADS1115 ADC.

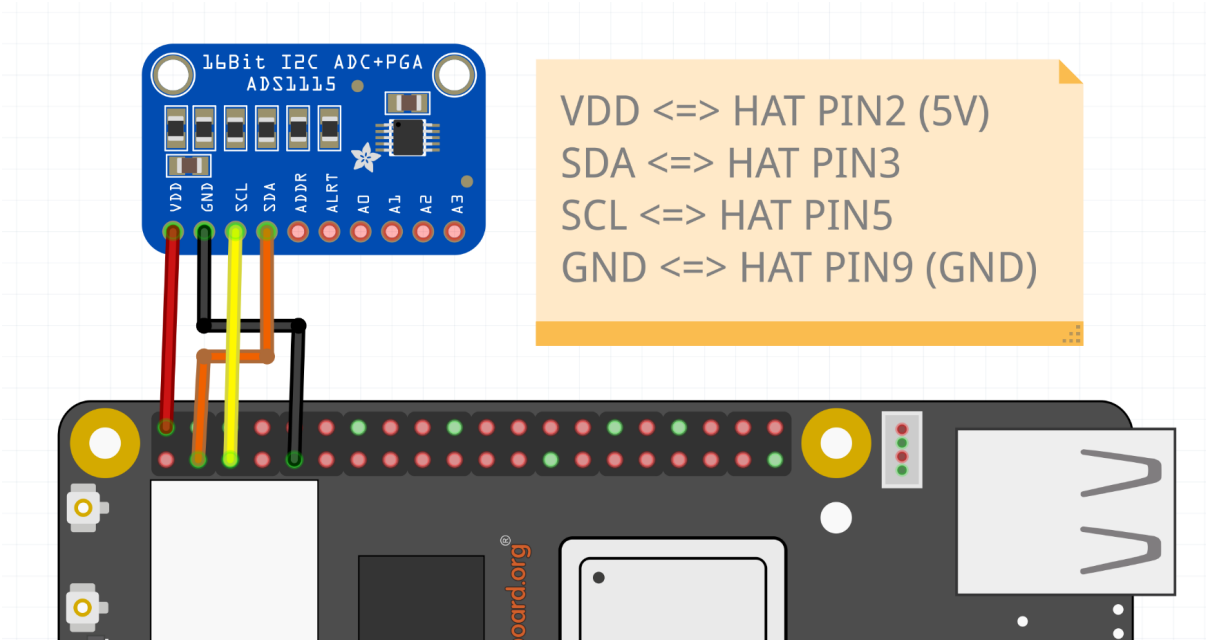


Fig. 5.26: ADS1115 connection

```
debian@BeagleBone:~$ i2cdetect -y -r 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                         -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- 48 -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

ADC Parameters

PGA The pga is the programmable gain amplifier (values are full scale), in the device tree overlay we have `ti,gain = <#>;` where # can be following,

- 0: +/- 6.144 V
- 1: +/- 4.096 V (default)
- 2: +/- 2.048 V
- 3: +/- 1.024 V
- 4: +/- 0.512 V
- 5: +/- 0.256 V

Data rate The data_rate in samples per second, in the device tree overlay we have `ti,datarate = <#>;` where # can be following,

- 0: 8
- 1: 16
- 2: 32

- 3: 64
- 4: 128
- 5: 250
- 6: 475
- 7: 860 (default)

ADC Inputs The inputs can be made available by 8 sysfs input files in0_input - in7_input,

- in0: Voltage over AIN0 and AIN1.
- in1: Voltage over AIN0 and AIN3.
- in2: Voltage over AIN1 and AIN3.
- in3: Voltage over AIN2 and AIN3.
- in4: Voltage over AIN0 and GND.
- in5: Voltage over AIN1 and GND.
- in6: Voltage over AIN2 and GND.
- in7: Voltage over AIN3 and GND.

Note: In the device tree overlay we have channel@4 – channel@7 device tree nodes representing in4 – in7 from the list above.

Using kernel driver

To use the kernel driver to drive the ADS1115 ADC, we have created an overlay /boot/firmware/overlays/k3-am67a-beagley-ai-i2c1-ads1115.dtbo. To load the overlay you have to add fdtoverlays /overlays/k3-am67a-beagley-ai-i2c1-ads1115.dtbo to /boot/firmware/extlinux/extlinux.conf as shown below.

```
...
...
...

label microSD (default)
kernel /Image
append console=ttyS2,115200n8 root=/dev/mmcblk1p3 ro rootfstype=ext4 resume=/
↪dev/mmcblk1p2 rootwait net.ifnames=0 quiet
fdtdir /
fdt /ti/k3-am67a-beagley-ai.dtb
fdtoverlays /overlays/k3-am67a-beagley-ai-i2c1-ads1115.dtbo
```

After rebooting the board you should see /sys/bus/iio/devices/iio:device0 available.

```
debian@BeagleBone:~$ ls /sys/bus/iio/devices/ | grep iio
iio:device0
```

To show all the *ADC Inputs* you can create a script called adcreader.sh.

- Create the file,

```
nano adcreader.sh
```

- Copy and paste the content below,

```
in0=$(cat /sys/bus/iio/devices/iio\:device0/in_voltage0-voltage1_raw)
in1=$(cat /sys/bus/iio/devices/iio\:device0/in_voltage0-voltage3_raw)
in2=$(cat /sys/bus/iio/devices/iio\:device0/in_voltage1-voltage3_raw)
in3=$(cat /sys/bus/iio/devices/iio\:device0/in_voltage2-voltage3_raw)
in4=$(cat /sys/bus/iio/devices/iio\:device0/in_voltage0_raw)
in5=$(cat /sys/bus/iio/devices/iio\:device0/in_voltage1_raw)
in6=$(cat /sys/bus/iio/devices/iio\:device0/in_voltage2_raw)
in7=$(cat /sys/bus/iio/devices/iio\:device0/in_voltage3_raw)

echo "in0=${in0}\nin1=${in1}\nin2=${in2}\nin3=${in3}\nin4=${in4}\nin5=${in5}\
↪nin6=${in6}\nin7=${in7}"
```

- To allow the execution of the script as normal user use the command below,

```
chmod +x adcreader.sh
```

- To view the ADC updates every 100ms use the `watch` command as shown below,

```
watch -n 0.1 adcreader.sh
```

The above command should show the values as shown below and it will update them every `0.1s`,

```
Every 0.1s: adcreader1.sh

in0=0
in1=-2
in2=2
in3=0
in4=4447
in5=4762
in6=4470
in7=4696
```

Todo: Add further testing steps, results, and images.

5.6 Using PCA9685 Motor Drivers

There are several such "Motor and Servo Driver HATs" available on Amazon, Adafruit and other marketplaces. While different manufacturers implement them slightly differently, the operating principle remains the same.

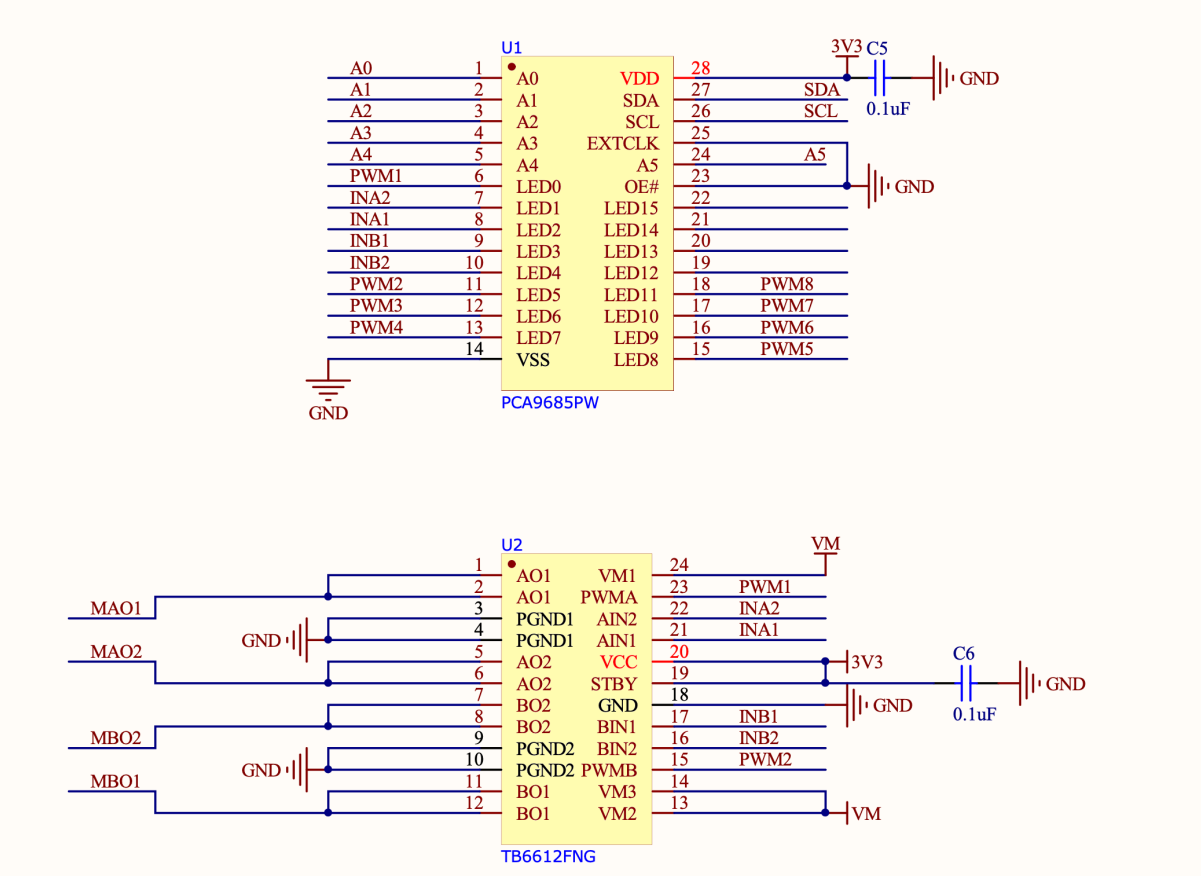
This guide aims to show you examples for two, namely the Xicoolee and Adafruit variants and how you can modify the example Python userspace library for other variations.

5.6.1 Operating Principle

The NXP PCA9685 is a simple 16-channel, 12-bit PWM controller that communicates over I2C.

While originally designed as an LED driver, it's ability to output PWM also makes it suitable as a Servo Motor driver.

In addition, to add the ability to drive DC motors, some board designers add one or two Toshiba TB6612FNG dual motor drivers as shown in the schematic below.



If we look at the Xicoolee board and compare it to the schematic, we see that indeed Servo Channels 3-8 on the PCB Silkscreen match pins 12 through 18 of the PWM Driver, while PWM1, PWM2, INA1/2 and INB1/2 are used in conjunction with the TB6612FNG.

Looking at the TB6612FNG Datasheet, we can see that the IN pins for Channels A and B (INAx, INBx) are used to control the direction or “mode” of the DC motor, while the PWM signal controls the rotation speed for that particular channel.

TOSHIBA TB6612FNG

H-SW Control Function

Input				Output		
IN1	IN2	PWM	STBY	OUT1	OUT2	Mode
H	H	H/L	H	L	L	Short brake
L	H	H	H	L	H	CCW
		L	H	L	L	Short brake
H	L	H	H	H	L	CW
		L	H	L	L	Short brake
L	L	H	H	OFF (High impedance)		Stop
H/L	H/L	H/L	L	OFF (High impedance)		Standby

Thus, we can use the decoder table above to infer that to drive motor channel A at 50% speed clockwise, we

would set the PCA9685 to output INA1 High, INA2 Low and PWM1 at a 50% duty cycle.

If we wanted to go counter-clockwise, we would simply swap things around so INA1 was Low, INA2 was High and assuming we want to keep the same rotation speed, PWM1 at a 50% duty cycle.

Lastly, we have the option for a "Short Brake" for the motors but please note that it is not recommended to keep motors in this state as that shorts the coils internally and will cause them to heat up over time. If you want to stop your motor, you should issue a "Short brake" state followed by a short delay to allow the motor to physically stop rotating and then leave the motor in the "Stop" state (which de-energizes the coils) by setting IN1 and IN2 to LOW.

But enough theory, let's use some actual code to make things spin...

5.6.2 Using Adafruit ServoKit

If you are looking to drive Servo motors accurately and not particularly interested in driving DC motors, you may consider using the Adafruit ServoKit library which simplifies this type of use case. As with all python modules, make sure you do so inside a virtual environment as shown below!

```
mkdir project-name && cd project-name
python3 -m venv .venv
source .venv/bin/activate
sudo pip3 install --upgrade setuptools
sudo pip install --upgrade adafruit-python-shell
wget https://raw.githubusercontent.com/adafruit/Raspberry-Pi-Installer-
↪Scripts/master/raspi-blinka.py
sudo python raspi-blinka.py
pip3 install adafruit-circuitpython-servokit adafruit-circuitpython-
↪busdevice adafruit-circuitpython-register
```

From here, you should be able to run some example code such as the following:

```
import time
from adafruit_servokit import ServoKit

# Set channels to the number of servo channels on your kit.
# 8 for FeatherWing, 16 for Shield/HAT/Bonnet.
kit = ServoKit(channels=16)

kit.servo[0].angle = 180
kit.continuous_servo[1].throttle = 1
time.sleep(1)
kit.continuous_servo[1].throttle = -1
time.sleep(1)
kit.servo[0].angle = 0
kit.continuous_servo[1].throttle = 0
```

To explore ServoKit further, check out the ServoKit Github Page and Examples

5.6.3 Python User-space Driver

As mentioned before, the PCA9685 is a rather simple I2C device, so the driver for it is equally simple: PCA9685.py

Simply download this to the root of your project and you are most of the way there.

From there, you simply need an import statement and to define the driver instance:

```
from PCA9685 import PCA9685

pwm = PCA9685(0x60, debug=False) #Default I2C Address for the shield is 0x60
pwm.setPWMFreq(50) #Most Servo Motors use a PWM Frequency of 50Hz
```

You can now drive LEDs or servo motors by issuing the following command (replacing pin and dutyCycle with your particular values):

```
pwm.setDutycycle(pin, dutyCycle)
```

5.6.4 WaveShare Motor and Servo Driver HAT

Waveshare writes some of the better documentation for these types of Motor Driver HATs

Todo: Add more information on Waveshare motor & servo driver HAT.

5.6.5 XICOOLEE Motor and Servo Driver HAT

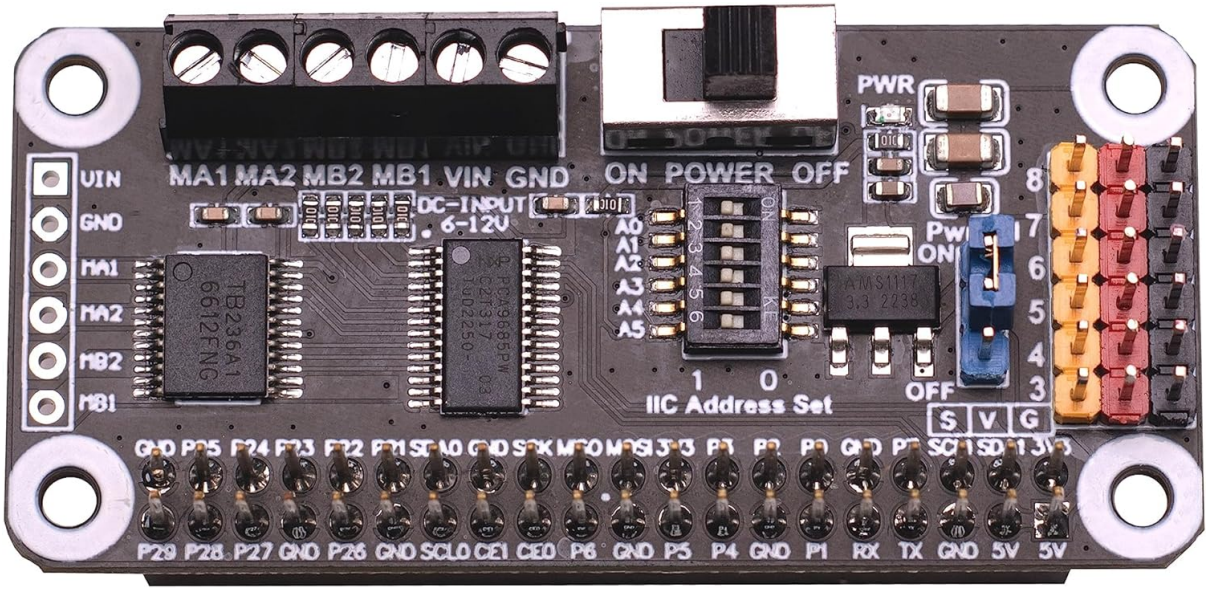


Photo Credit - Xicoolee

Looking at the schematic for the Xicoolee HAT, we see that we need to define our DC motor pins as follows:

```
#Xicoolee TB6612FNG

self.PWMA = 0
self.AIN1 = 2
self.AIN2 = 1
self.PWMB = 5
self.BIN1 = 3
self.BIN2 = 4
```

We can then run some simple example code as shown below:

```
#!/usr/bin/python

from PCA9685 import PCA9685
import time

Dir = [
    'forward',
    'backward',
]
```

(continues on next page)

(continued from previous page)

```
pwm = PCA9685(0x40, debug=False)
pwm.setPWMFreq(50)

class MotorDriver():
    def __init__(self):
        # Match these to your particular HAT!
        self.PWMA = 0
        self.AIN1 = 2
        self.AIN2 = 1
        self.PWMB = 5
        self.BIN1 = 3
        self.BIN2 = 4

    def MotorRun(self, motor, index, speed):
        if speed > 100:
            return
        if(motor == 0):
            pwm.setDutycycle(self.PWMA, speed)
            if(index == Dir[0]):
                print ("1")
                pwm.setLevel(self.AIN1, 0)
                pwm.setLevel(self.AIN2, 1)
            else:
                print ("2")
                pwm.setLevel(self.AIN1, 1)
                pwm.setLevel(self.AIN2, 0)
        else:
            pwm.setDutycycle(self.PWMB, speed)
            if(index == Dir[0]):
                print ("3")
                pwm.setLevel(self.BIN1, 0)
                pwm.setLevel(self.BIN2, 1)
            else:
                print ("4")
                pwm.setLevel(self.BIN1, 1)
                pwm.setLevel(self.BIN2, 0)

    def MotorStop(self, motor):
        if (motor == 0):
            pwm.setDutycycle(self.PWMA, 0)
        else:
            pwm.setDutycycle(self.PWMB, 0)

print("this is a motor driver test code")
Motor = MotorDriver()

print("forward 2 s")
Motor.MotorRun(0, 'forward', 100)
Motor.MotorRun(1, 'forward', 100)
time.sleep(2)

print("backward 2 s")
Motor.MotorRun(0, 'backward', 100)
Motor.MotorRun(1, 'backward', 100)
time.sleep(2)

print("stop")
Motor.MotorStop(0)
Motor.MotorStop(1)
```

5.6.6 Adafruit DC & Stepper Motor HAT

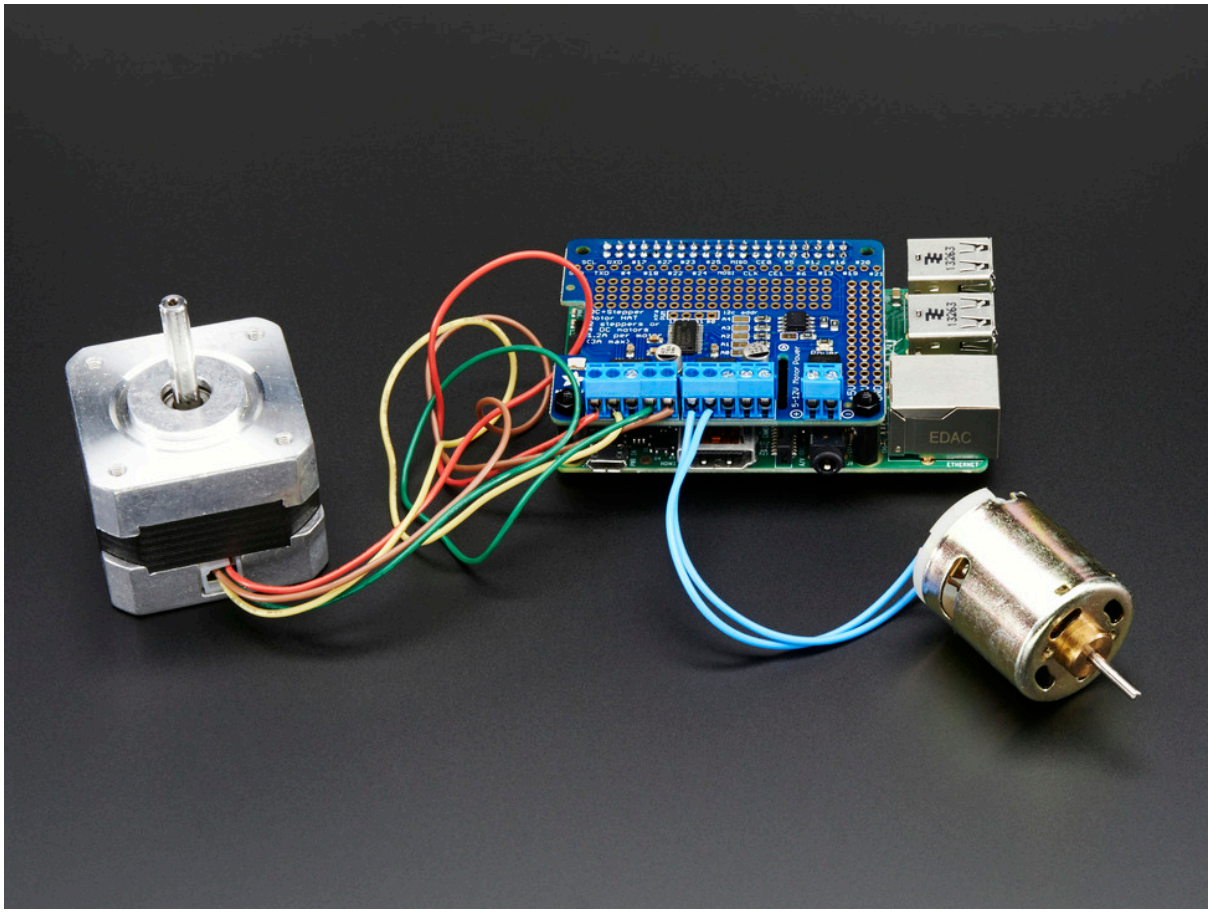


Photo Credit - Adafruit

Looking at the schematic for the Adafruit HAT, we see that we need to define our DC motor pins as follows:

```
#Adafruit TB6612FNG #1

self.PWMA = 8
self.AIN1 = 10
self.AIN2 = 9
self.PWMB = 13
self.BIN1 = 11
self.BIN2 = 12

#Adafruit TB6612FNG #2

self.PWMA_2 = 2
self.AIN1_2 = 4
self.AIN2_2 = 3
self.PWMB_2 = 7
self.BIN1_2 = 5
self.BIN2_2 = 6
```

Todo: Expand on running 2 DC motor objects

5.7 Booting from NVMe Drives

Todo: Add further testing steps, results, and images.

BeagleY-AI supports a PCI-Express x1 interface which enables data rates of up to 1GB/s for high speed expansion.

Note: While the SoC supports PCI-e Gen 3, the flat-flex connector required by HATs is only rated for PCI-e Gen 2, so, as is the case with other similar boards in this form factor, actual transfer speeds may be limited to Gen 2, depending on a variety of layout and environmental factors

This enables it to take advantage of standard PC NVMe drives which offer exponentially higher random and sequential read/write speeds as well as improved endurance over SD cards or traditional eMMC storage.

While the boot-ROM on the AM67 SoC does not support direct boot-to-NVMe, we can use a method where we boot U-Boot from the SD Card and then use it to load the Linux filesystem from external NVMe storage.

5.7.1 Verified HATs and Drives

Most/All HATs and NVMe drives should work, but the following have been verified to work as part of writing this guide:

HATs:

1. Geekworm X1001 PCIe to M.2 Key-M
2. Geekworm X1000 PCIe M.2 Key-M

NVMe drives:

1. Kingston OM3PDP3512B (512GB 2230)
2. Kingston NV2 (512GB 2280)

Drive Adapters (3D Printable):

The X1000 above uses the slightly uncommon 2242 drive size, so, an adapter may be required to mount a 2230 drive.

1. A simple adapter from `@eliasjonsson` on Printables works great - https://www.printables.com/model/578236-m2-ssd-2230-to-2242
2. Similar adapters exist for 2230 to 2280 for example such as this one from `@nzalog` - https://www.printables.com/model/217264-2230-to-2280-m2-adapter-ssd

5.7.2 Step by step

Note: This article was written using the BeagleY-AI Debian XFCE 12.5 2024-06-12 image.

Step 1. Boot from SD Normally

Grab the latest BeagleY-AI SD Image from (BeagleBoard.org/distros.)

Once logged in and at the terminal, make sure your system is up to date (a reboot is also recommended after updating)

```
sudo apt-get update && sudo apt-get full-upgrade -y
sudo reboot
```

Step 2. Verify that your NVMe drive is detected

The command `lspci` will list the attached PCI Express devices on the system:

```
debian@BeagleY:~$ lspci
```

You should see an output similar to the following, where the first entrance is the SoC internal PCI Express bridge device and the second device listed is your NVMe drive, in this case, a Kingston OM3PDP3 drive.

```
00:00.0 PCI bridge: Texas Instruments Device b010
01:00.0 Non-Volatile memory controller: Kingston Technology Company, Inc.␣
↪OM3PDP3 NVMe SSD (rev 01)
```

Now that we know the PCIe device is detected, let's see if it's recognized as a Storage Device:

The command `lsblk` will list the attached storage devices on the system:

```
debian@BeagleY:~$ lsblk
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
mmcblk1     179:0    0 29.7G  0 disk
├─mmcblk1p1 179:1    0  256M  0 part /boot/firmware
├─mmcblk1p2 179:2    0    4G  0 part [SWAP]
└─mmcblk1p3 179:3    0 25.5G  0 part /
nvme0n1     259:0    0 476.9G  0 disk
└─nvme0n1p1 259:1    0 476.9G  0 part
```

Here we see that two devices are connected, `mmcblk1` corresponds to our SD card, and `nvme0n1` corresponds to our NVMe drive, so everything is ready to go!

If your drives aren't listed as expected, please check the Troubleshooting section at the end of this document.

Step 3a. Transfer your root filesystem over to NVMe with a bootmenu option (recommended)

For this method, you will need Raspberry Pi Debug Probe or similar serial (USB to UART) adapter, to select the `2: transfer microSD rootfs to NVMe (advanced)` the boot menu option:

```
Scanning mmc 1:1...
Found /extlinux/extlinux.conf
Retrieving file: /extlinux/extlinux.conf
BeagleY-AI microSD (extlinux.conf) (swap enabled)
1:  microSD (production test)
2:  transfer microSD rootfs to NVMe (advanced)
3:  microSD (debug)
4:  microSD (default)
Enter choice: 2
```

The BeagleY-AI will shutdown when complete

Step 3b. Transfer your root filesystem over to NVMe with a shell script

A variety of useful scripts are available in `/opt/`, one of them enables us to move our micro-sd contents to NVMe and make BeagleY-AI boot from there directly.

The following 3 commands will change your U-boot prompt to boot from NVMe by default, but the serial boot menu will still enable you to fall back to SD boot or other modes if something happens.

Note: This will copy the entire contents of your SD card to the NVMe drive, so expect it to take upwards of 15 minutes. This only needs to be run one time

```
sudo cp -v /opt/u-boot/bb-u-boot-beagley-ai/beagley-ai-microsd-to-nvme-w-
↪swap /etc/default/beagle-flasher
```

```
sudo beagle-flasher-mv-rootfs-to-nvme
```

```
sudo reboot
```

Step 4. (optional) Verify u-boot (Serial Debug) will jump to the rootfs on the NVMe

Note: boot menu `1: return to microSD` will allow you to return to the microSD rootfs for any reasons.

```
Scanning mmc 1:1...
Found /extlinux/extlinux.conf
Retrieving file: /extlinux/extlinux.conf
BeagleY-AI NVMe (extlinux.conf) (swap enabled)
1:  return to microSD
2:  NVMe (debug)
3:  NVMe (default)
Enter choice: 3
```

Enjoy NVMe speeds!

Now that we've run the scripts above, you should see that lsblk now reports that our `/` or root filesystem is on the `nvme0n1p1` partition, meaning we are successfully booting from the NVMe drive.

It's subtle, but the change can be seen by running `lsblk` again.

```
debian@BeagleY:~$ lsblk
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
mmcblk1     179:0    0  29.8G  0 disk
├─mmcblk1p1 179:1    0   256M  0 part /boot/firmware
├─mmcblk1p2 179:2    0     4G  0 part
└─mmcblk1p3 179:3    0  25.6G  0 part
nvme0n1     259:0    0 931.5G  0 disk
├─nvme0n1p1 259:1    0     4G  0 part [SWAP]
└─nvme0n1p2 259:2    0 927.5G  0 part /
```

Congratulations!

5.7.3 Troubleshooting

While most setups should work, it is possible that a combination of Software, Hardware or both can result in minor issues. Here are some ideas for troubleshooting on your own:

Check that your cables are plugged in and oriented correctly

The flat-flex ribbon cable will only connect correctly one way, so ensure the orientation is correct with your expansion HAT manual and that the ribbon cable is correctly seated.

A note on power-hungry drives

While most drives can be powered as-is with only the ribbon cable, some drives, especially high end full-size 2280 drives may consume more power than normal for an M.2 connector. For such cases, some HAT expansions will provide a means of providing external supplemental power. If your drive is not detected, it may be worthwhile to try using a drive from a different manufacturer as a troubleshooting step.

As a side note, since 2230 drives are normally designed to run in Laptops, they tend to also consume less power than their desktop counterparts and as such, are a "safer" option.

Check the Linux Kernel Logs for PCI:

You should see something similar to below without further errors:

```
debian@BeagleY:~$ dmesg | grep "PCI"
[    0.005276] PCI/MSI: /bus@f0000/interrupt-controller@1800000/msi-
↪controller@1820000 domain created
[    0.158546] PCI: CLS 0 bytes, default 64
[    3.674209] j721e-pcie-host f102000.pcie: PCI host bridge to bus 0000:00
[    3.742406] pci 0000:01:00.0: 7.876 Gb/s available PCIe bandwidth,␣
↪limited by 8.0 GT/s PCIe x1 link at 0000:00:00.0 (capable of 31.504 Gb/s␣
↪with 8.0 GT/s PCIe x4 link)
[    4.915630] pci 0000:00:00.0: PCI bridge to [bus 01]
```

Still having issues?

Post questions on the forum under the tag "beagley-ai".

5.8 Using IMX219 CSI Cameras

Todo: Add further testing steps, results, and images.

5.8.1 Camera connection

You have to make sure the CSI camera FPC cable is connected in proper orientation for camera to work properly. The image below shows how it should look when connected correctly.

Tip: The image shows camera connection to `CSI0` port, you can similarly connect a camera to `CSI1/DSI0` port

5.8.2 Configuring CSI camera

To enable an IMX219 CSI camera, we have to modify the following file: `/boot/firmware/extlinux/extlinux.conf`. We can check the available list of Device Tree Overlays using command below,

```
ls /boot/firmware/overlays/ | grep "beagley"
```

When executed the above command should give output show below,

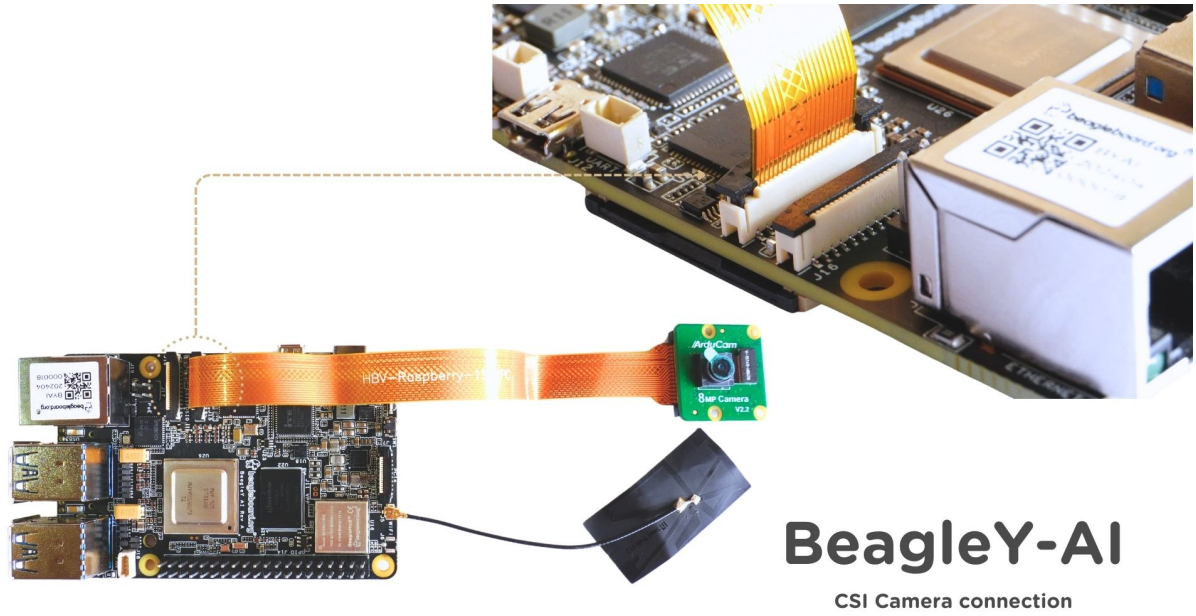


Fig. 5.27: BeagleY-AI camera connection

```
debian@BeagleBone:~$ ls /boot/firmware/overlays/ | grep "beagley"
k3-am67a-beagley-ai-csi0-imx219.dtbo
k3-am67a-beagley-ai-csi0-ov5640.dtbo
k3-am67a-beagley-ai-csi1-imx219.dtbo
k3-am67a-beagley-ai-dsi-rpi-7inch-panel.dtbo
k3-am67a-beagley-ai-lincolntech-185lcd-panel.dtbo
```

Using CSI Port 0

Add the line shown below to `/boot/firmware/extlinux/extlinux.conf` file to load the IMX219 CSI0 device tree overlay:

```
fdtoverlays /overlays/k3-am67a-beagley-ai-csi0-imx219.dtbo
```

Your `/boot/firmware/extlinux/extlinux.conf` file should look something like this:

```
label microSD (default)
   kernel /Image
   append console=ttyS2,115200n8 root=/dev/mmcblk0p2 ro rootfstype=ext4␣
↪rootwait net.ifnames=0
   fdtdir /
   fdt /ti/k3-am67a-beagley-ai.dtb
   fdtoverlays /overlays/k3-am67a-beagley-ai-csi0-imx219.dtbo
   initrd /initrd.img
```

Now reboot you BeagleY-AI to load the overlay,

```
sudo reboot
```

After `reboot` you can use `beagle-camera-setup` to setup your IMX219 CSI camera,

```
sudo beagle-camera-setup
```

`beagle-camera-setup` should give you below output,

```
debian@beagle:~$ sudo beagle-camera-setup
[sudo] password for beagle:
IMX219 Camera 0 detected
   device = /dev/video-imx219-cam0
   name = imx219
   format = [fmt:SRGGB8_1X8/1920x1080]
   subdev_id = /dev/v4l-imx219-subdev0
   isp_required = yes
```

To check if the configuration is successfull you can check the video devices with `ls /dev/ | grep video` and you should see `video-imx219-cam0` listed as show below,

```
beagle@beagle:~$ ls /dev/ | grep video
video-imx219-cam0
video0
video1
video2
video3
video4
video5
video6
video7
video8
```

5.8.3 Using CSI Port 1

Todo: add instructions to setup CSI1

5.8.4 Photos & video

Todo: add instruction to take photos and videos

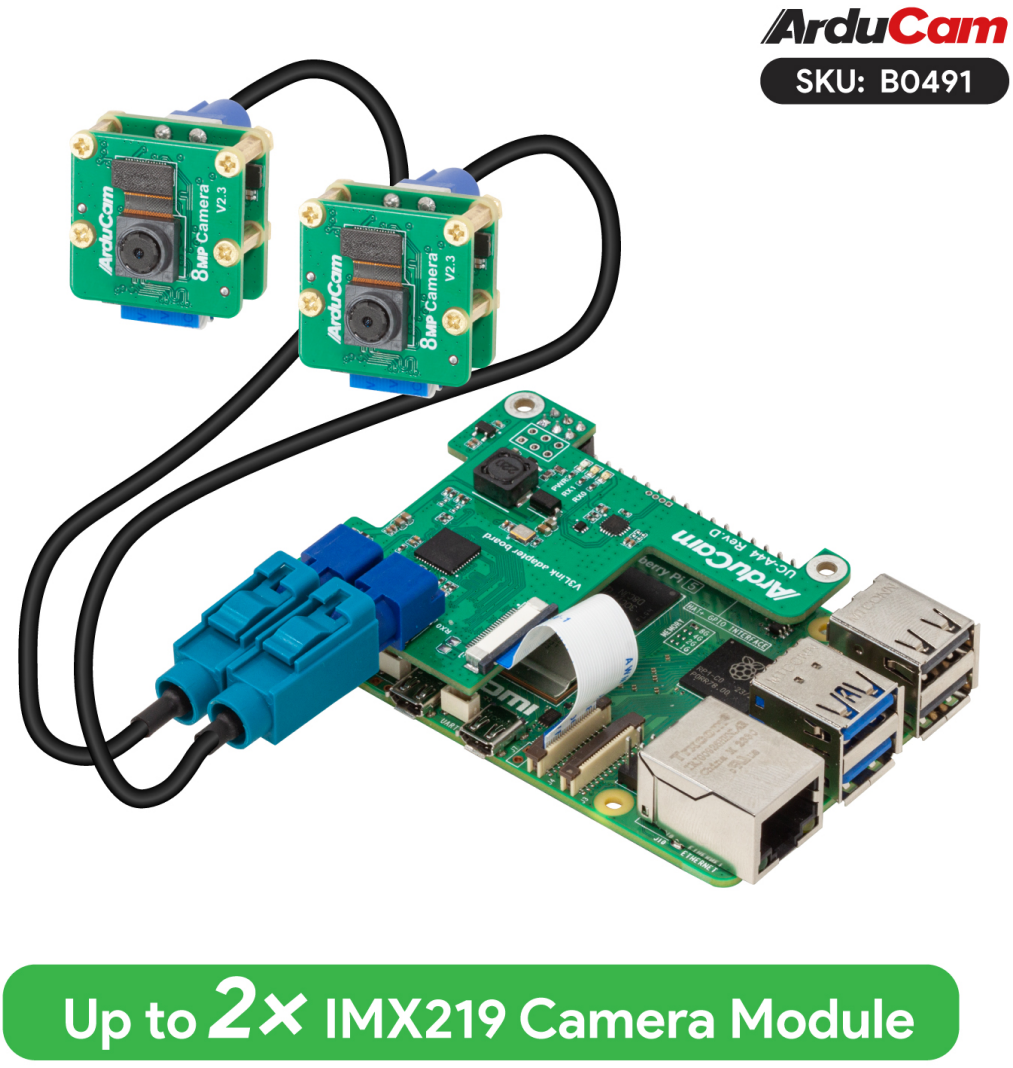
5.8.5 Troubleshooting

```
Found /extlinux/extlinux.conf
Retrieving file: /extlinux/extlinux.conf
beagley-ai microSD (extlinux.conf)
   1:      microSD Recovery
   2:      microSD (RPI 7inch panel)
   3:      microSD (lincolntech-1851cd panel)
   4:      microSD (csi0 imx219)
   5:      microSD (csi1 imx219)
   6:      microSD (csi0 ov5640)
   7:      microSD (default)
Enter choice: 4
   4:      microSD (csi0 imx219)
```

5.9 Using the Arducam Dual V3Link Camera Kit

Todo: Add further testing steps, results, and images.

The Arducam Dual V3Link Camera Kit is an IMX219 based kit that leverages Texas Instruments' FPDLink technology to enable using two CSI cameras over a single port up to 15 meters away using twisted pair cables.



Note: Unlike the larger quad-camera kit, the dual camera kit aims to simplify the software stack and improve interoperability with the Raspberry Pi and other non-TI SBCs by forgoing the ability to support multi-stream CSI inputs. This means that it is limited to "switching" between the two FPDLink inputs but has the benefit of not requiring additional drivers beyond support for the base CSI camera driver (IMX219 in this case)

5.9.1 Initial Hardware Connection

Simply plug in the HAT into the BeagleY GPIO header and connect the CSI header as shown below.

Either CSI header may be connected but make sure you use the corresponding CSI port DTS when enabling your “camera”.

Todo: ADD CSI 0/1 Header Location photo.

5.9.2 Verify that the HAT is connected

The Arducam HAT should present itself as an I2C device on Bus 1.

To check that the I2C Bus looks like we expect:

```
sudo i2cdetect -r -y 1
```

To verify actual communication with the FPDlink device, we issue the following command:

```
sudo i2ctransfer -f -y 4 w3@0x0c 0xff 0x55 0x01 r1
```

5.9.3 Switching CSI Channels

The channel numbering for FPDLink goes from 1 to 2 (as opposed to counting from 0 as is the case for CSI)

Thus, to select video output from channel 1:

```
sudo i2ctransfer -f -y 4 w3@0x0c 0xff 0x55 0x01
```

To switch to channel 2:

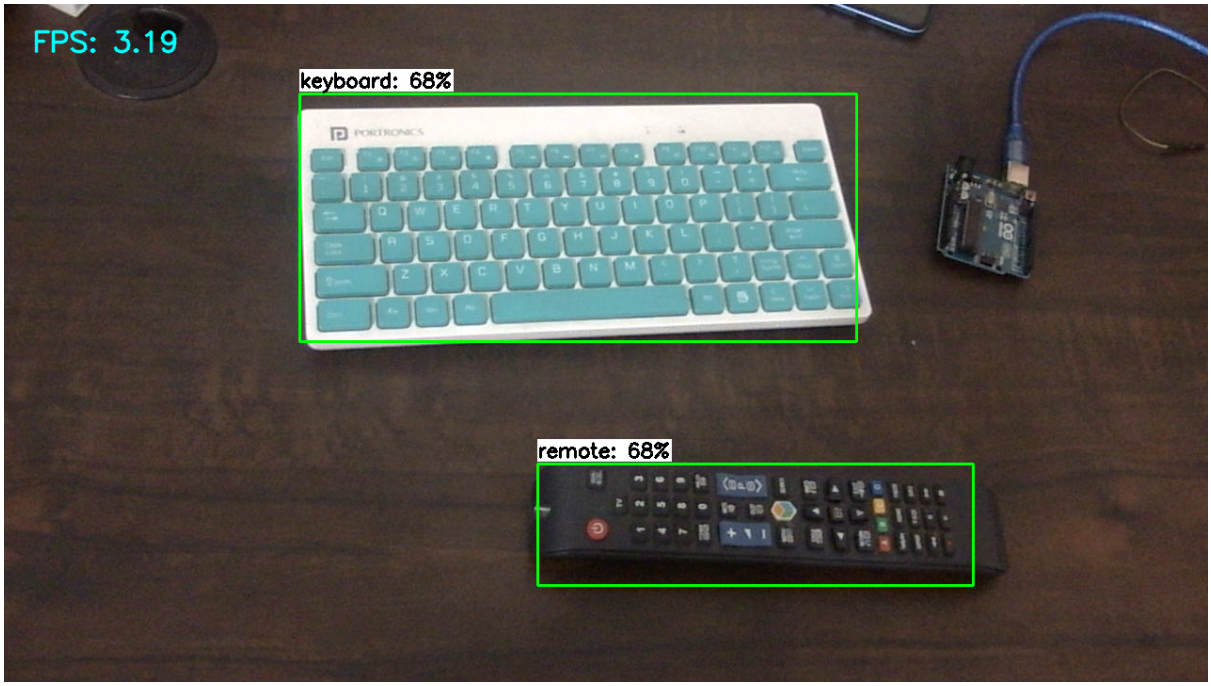
```
sudo i2ctransfer -f -y 4 w3@0x0c 0xff 0x55 0x02
```

5.9.4 Troubleshooting

For additional documentation and support, see the Arducam Docs.

5.10 TensorFlow Lite Object Detection

This document describes how to set up and run an object detection model using TensorFlow Lite on the BeagleY-AI platform. Below is a demonstration.



To run the object detection model on the BeagleY-AI, you will need the following:

- **BeagleY-AI Board**: Make sure to refer to the *BeagleY-AI standalone connection* for proper setup.
- **USB Webcam**: The model has been tested with the Logitech Webcam C270, but it should work well with other webcam too.
- **Active Internet Connection**: Necessary for the installation of modules. Please check the *WiFi connection guide* for setting up the network.

5.10.1 Step 1: Installation of Conda

In this step, we will install a lightweight version of Conda.

```
wget https://github.com/conda-forge/miniforge/releases/download/24.3.0-0/
↪Mambaforge-24.3.0-0-Linux-aarch64.sh
bash Mambaforge-24.3.0-0-Linux-aarch64.sh
```

After accepting the license terms you can verify the installation by

```
conda --version
```

5.10.2 Step 2: Create Virtual Environment

Create a virtual environment with Python 3.9.

```
conda create --name myenv python=3.9
```

5.10.3 Step 3: Activate the Virtual Environment

Activate the virtual environment created in the previous step.

```
conda activate myenv
```

5.10.4 Step 4: Install Necessary Modules

Install the required Python modules.

```
pip install https://github.com/google-coral/pycoral/releases/download/v2.0.0/
↪tflite_runtime-2.5.0.post1-cp39-cp39-linux_aarch64.whl
pip install numpy==1.26.4
pip install opencv-python
```

5.10.5 Step 5: Load Necessary Pretrained Models

Create a directory for the object recognition models and download a pretrained model.

```
mkdir object-recognition
cd object-recognition
wget https://storage.googleapis.com/download.tensorflow.org/models/tflite/
↪coco_ssd_mobilenet_v1_1.0_quant_2018_06_29.zip
unzip coco_ssd_mobilenet_v1_1.0_quant_2018_06_29.zip -d TFLite_model
```

Tip: You can train your own model using TensorFlow Lite.Here are some resources

1. Train TensorFlow Lite Object Detection Model.
2. TensorFlow Lite Model Maker.

5.10.6 Step 6: Connect Your USB Webcam

Connect your USB webcam via a USB socket.

```
ls -l /dev | grep video
```

```
(myenv) debian@BeagleBone:~/object-recognition$ ls -l /dev | grep video
crw-rw----  1 root video    29,   0 Aug  9 09:55 fb0
crw-rw----+ 1 root video   234,   0 Aug 10 06:42 media0
crw-rw----+ 1 root video    81,   0 Aug  9 09:55 video0
crw-rw----+ 1 root video    81,   1 Aug  9 09:55 video1
crw-rw----+ 1 root video    81,   2 Aug  9 09:55 video2
crw-rw----+ 1 root video    81,   3 Aug 10 06:42 video3
crw-rw----+ 1 root video    81,   4 Aug 10 06:42 video4
```

Note: Check the video driver with the above command. Here its 3 in my case.

5.10.7 Step 7: Create the Code File

Create a Python file for running object detection.

```
nano object-detection.py
```

Paste the following code into the file:

```
import os
import argparse
import cv2
import numpy as np
```

(continues on next page)

(continued from previous page)

```
import time
from threading import Thread
import importlib.util
from typing import List
import sys
from tflite_runtime.interpreter import Interpreter, load_delegate

video_driver_id = 3

class VideoStream:
    """Handles video streaming from the webcam."""
    def __init__(self, resolution=(640, 480), framerate=30):
        self.stream = cv2.VideoCapture(video_driver_id)
        self.stream.set(cv2.CAP_PROP_FOURCC, cv2.VideoWriter_fourcc(*'MJPG'))
        self.stream.set(3, resolution[0])
        self.stream.set(4, resolution[1])
        self.grabbed, self.frame = self.stream.read()
        self.stopped = False

    def start(self):
        """Starts the thread that reads frames from the video stream."""
        Thread(target=self.update, args=()).start()
        return self

    def update(self):
        """Continuously updates the frame from the video stream."""
        while True:
            if self.stopped:
                self.stream.release()
                return
            self.grabbed, self.frame = self.stream.read()

    def read(self):
        """Returns the most recent frame."""
        return self.frame

    def stop(self):
        """Stops the video stream and closes resources."""
        self.stopped = True

def load_labels(labelmap_path: str) -> List[str]:
    """Loads labels from a label map file."""
    try:
        with open(labelmap_path, 'r') as f:
            labels = [line.strip() for line in f.readlines()]
        if labels[0] == '???':
            labels.pop(0)
        return labels
    except IOError as e:
        print(f"Error reading label map file: {e}")
        sys.exit()

def main():
    # Argument parsing
    parser = argparse.ArgumentParser()
    parser.add_argument('--modeldir', required=True, help='Folder the .
↪tflite file is located in')
    parser.add_argument('--graph', default='detect.tflite', help='Name of␣
↪the .tflite file')
    parser.add_argument('--labels', default='labelmap.txt', help='Name of␣
↪the labelmap file')
```

(continues on next page)

(continued from previous page)

```
    parser.add_argument('--threshold', default='0.5', help='Minimum␣
↪confidence threshold')
    parser.add_argument('--resolution', default='1280x720', help='Desired␣
↪webcam resolution')
    args = parser.parse_args()

    # Configuration
    model_path = os.path.join(os.getcwd(), args.modeldir, args.graph)
    labelmap_path = os.path.join(os.getcwd(), args.modeldir, args.labels)
    min_conf_threshold = float(args.threshold)
    resW, resH = map(int, args.resolution.split('x'))

    # Load labels and interpreter
    labels = load_labels(labelmap_path)
    interpreter = Interpreter(model_path=model_path)
    interpreter.allocate_tensors()

    # Get model details
    input_details = interpreter.get_input_details()
    output_details = interpreter.get_output_details()
    height, width = input_details[0]['shape'][1:3]
    floating_model = (input_details[0]['dtype'] == np.float32)

    outname = output_details[0]['name']
    boxes_idx, classes_idx, scores_idx = (1, 3, 0) if
↪'StatefulPartitionedCall' in outname else (0, 1, 2)

    # Initialize video stream
    videostream = VideoStream(resolution=(resW, resH), framerate=30).start()
    time.sleep(1)

    frame_rate_calc = 1
    freq = cv2.getTickFrequency()

    while True:
        t1 = cv2.getTickCount()
        frame = videostream.read()
        frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        frame_resized = cv2.resize(frame_rgb, (width, height))
        input_data = np.expand_dims(frame_resized, axis=0)

        if floating_model:
            input_data = (np.float32(input_data) - 127.5) / 127.5

        interpreter.set_tensor(input_details[0]['index'], input_data)
        interpreter.invoke()

        boxes = interpreter.get_tensor(output_details[boxes_idx]['index'])[0]
        classes = interpreter.get_tensor(output_details[classes_idx]['index
↪'])[0]
        scores = interpreter.get_tensor(output_details[scores_idx]['index
↪'])[0]

        for i in range(len(scores)):
            if min_conf_threshold < scores[i] <= 1.0:
                ymin, xmin, ymax, xmax = [int(coord) for coord in (boxes[i]␣
↪* [resH, resW, resH, resW])]
                cv2.rectangle(frame, (xmin, ymin), (xmax, ymax), (10, 255,␣
↪0), 2)
                object_name = labels[int(classes[i])]
                label = f'{object_name}: {int(scores[i] * 100)}%'
```

(continues on next page)

(continued from previous page)

```
                labelSize, baseLine = cv2.getTextSize(label, cv2.FONT_
↪HERSHEY_SIMPLEX, 0.7, 2)
                label_ymin = max(ymin, labelSize[1] + 10)
                cv2.rectangle(frame, (xmin, label_ymin - labelSize[1] - 10),␣
↪(xmin + labelSize[0], label_ymin + baseLine - 10), (255, 255, 255), cv2.
↪FILLED)
                cv2.putText(frame, label, (xmin, label_ymin - 7), cv2.FONT_
↪HERSHEY_SIMPLEX, 0.7, (0, 0, 0), 2)

        cv2.putText(frame, f'FPS: {frame_rate_calc:.2f}', (30, 50), cv2.FONT_
↪HERSHEY_SIMPLEX, 1, (255, 255, 0), 2, cv2.LINE_AA)
        cv2.imshow('Object detector', frame)

        t2 = cv2.getTickCount()
        time1 = (t2 - t1) / freq
        frame_rate_calc = 1 / time1

        if cv2.waitKey(1) == ord('q'):
            break

    cv2.destroyAllWindows()
    videostream.stop()

if __name__ == "__main__":
    main()
```

Note: Make sure to change your video driver ID depending on your video driver. Here, the video driver ID is set to 3.

5.10.8 Step 8: Run the Object Detection Script

To run the object detection script, use the following command. Replace *TFLite_model* with the path to your model directory if it differs:

```
python3 object_detection.py --modeldir=TFLite_model
```

A window will open, displaying the object detection model in action.

Chapter 6

Support

All support for BeagleY-AI design is through BeagleBoard.org community at BeagleBoard.org forum.

6.1 Production board boot media

Todo: Add production boot media link in `_static/epilog/production.image` and reference it here.

6.2 Certifications and export control

6.2.1 Export designations

- HS: 8471504090
- US HS: 8543708800
- UPC: 640265311062
- EU HS: 8471707000
- COO: CHINA

6.2.2 Size and weight

- Bare board dimensions: 85 x 56 x 20 mm
- Bare board weight: 50 g
- Full package dimensions: 140 x 100 x 40 mm
- Full package weight: 110g

6.3 Additional documentation

6.3.1 Hardware docs

For any hardware document like schematic diagram PDF, EDA files, issue tracker, and more you can checkout the BeagleY-AI design repository.

6.3.2 Software docs

For BeagleY-AI specific software projects you can checkout all the BeagleY-AI project repositories group.

6.3.3 Support forum

For any additional support you can submit your queries on our forum, https://forum.beagleboard.org/tag/beagley-ai

6.3.4 Pictures

6.4 Change History

Note: This section describes the change history of this document and board. Document changes are not always a result of a board change. A board change will always result in a document change.

6.4.1 Board Changes

For all changes, see https://openbeagle.org/beagley-ai/beagley-ai. Versions released into production are noted below.

Table 6.1: BeagleY-AI board change history

Rev	Changes	Date	By